Hochdynamische logische Netzwerke als Infrastruktur mobiler Agentensyteme

Verteilung, Aktualität und Skalierbarkeit

Dissertation

zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)

vorgelegt dem Rat der Fakultät für Mathematik und Informatik
der Friedrich-Schiller-Universität Jena

von
Dipl.-Wirt.-Ing. (FH) Arndt Döhler
geboren am 13. Januar 1971 in Apolda

Gutachter

1. Prof. Dr. Wilhelm R. Rossak

2. Prof. Dr. Ilka Philippow, TU Ilmenau

3. Prof. Dipl.-Ing. Erich Stein

Tag der letzten Prüfung des Rigorosums: 23.01.2008

Tag der öffentlichen Verteidigung: 08.02.2008

Kurzfassung

Mobile Agentensysteme können als ein neues Paradigma für verteilte Systeme und verteilte Kommunikation betrachtet werden. Sie spannen logische Netzwerke zwischen ihren im Internet verteilten Plattformen auf, welche als Infrastruktur dienen. In mobilen Agentensystemen der Stufe 2 sind die Agenten dabei sogar befähigt, sich selbst unter Nutzung bestimmter Routingdienste eine serviceorientierte Reiseroute über ihre Infrastruktur zu planen und entsprechend dieser durch das Netzwerk zu migrieren. Zur sinnvollen Planung sind aktuelle Umgebungsinformationen aus ihrer global verteilten Infrastruktur notwendig. Heutige Netzwerke sind allerdings durch die Verwendung von mobilen Endgeräten hochdynamisch geworden. Daher skaliert, unter Beachtung der für mobile Agentensysteme der Stufe 2 notwendigen Zeitschranken für aktuelle Dienstinformationen, die Frequenz und die Anzahl von nötigen Aktualisierungsnachrichten in globalen logischen Netzwerken nicht mit der Menge der möglichen Dienstangebote.

Diese Arbeit konzentriert sich daher auf die Verteilung und Aktualität von Dienstinformationen in einem solchen logischen Netzwerk. Sie berücksichtigt dabei auch die Zeitschranken, die von mobilen Endgeräten hervorgerufen werden, und die globale Skalierbarkeit von Dienstinformationen. Zuerst wird eine Untersuchung vorhandener Systeme vorgenommen und deren Defizite im Hinblick auf die Anforderungen mobiler Agentensysteme der Stufe 2 eruiert. Aus den Anforderungen werden gewünschte Systemeigenschaften abgeleitet, für welche ein allgemeines Lösungsmodell in Form einer universellen Softwarearchitektur, bestehend aus drei Komponenten, vorgestellt wird. Die Umsetzung der Lösungsideen und der Softwarearchitektur erfolgt dann im prototypischen Infrastrukturdienst-Framework *QuickLinkNet*, welches aus den drei Softwarekomponenten *QuickLink*, *ServiceJuggler* und *APLICOOVER* besteht.

QuickLinkNets Komponenten bauen ein transparentes, zweischichtiges logisches Netzwerk auf, welches als serviceorientierte Infrastruktur für mobile Agentensysteme der Stufe 2 dient, aber auch für andere verteilte Anwendungen in hochdynamischen Netzwerken verwendet werden kann. Die zwei Schichten des logischen Netzwerkes entsprechen zwei unterschiedlichen Sichten auf dieses: Eine lokale Sicht und eine globale Sicht. Jede Sicht wird über eine eigene Verteilungsstruktur realisiert und betont daher andere Systemeigenschaften. So hilft die lokale Sicht, umgesetzt durch QuickLink und ServiceJuggler, eine hohe Netzwerkdynamik und ein hohes Aktualisierungsniveau der Dienstinformationen in einem begrenzten Netzwerkbereich zu beherrschen. Dazu bedienen sich die Komponenten auch Mechanismen zur selbstadaptierenden Verwaltung des Netzwerkes, um die Auswirkungen der Netzwerkdynamik vor den verteilten Anwendungen als Nutzer der Infrastruktur zu verbergen. Durch das hochdynamische Management der Anwendungsdienstinformationen auf der lokalen Netzwerkschicht kann die globale Schicht, durch APLICOOVER realisiert, wiederum für die globale Skalierbarkeit der Dienstinformationen unter gemäßigteren Zeitschranken sorgen. Mit zahlreichen Tests und Messungen an den Komponenten QuickLinkNets in realen Netzwerkumgebungen werden die Vorteile des neuen Infrastrukturdienst-Frameworks QuickLinkNet nachgewiesen.

Abstract

Mobile agent systems may be regarded as a new paradigm to develop and support fully distributed systems and distributed communication. They are able to create a logical network in the internet, based on the platforms that support their execution, which in turn is utilized as an infrastructure by the mobile agents that comprise an application system. In so-called level 2 mobile agent systems, agents are even able to plan their route through that network themselves, based on relevant service descriptions and by using dedicated routing services. By migrating along such a route and making the appropriate, application specific decisions they are able to navigate the network in a highly autonomous fashion. However, for a really useful autonomous navigation, a very large number of up-to-date service descriptions must be handled in a very efficient way. This turns out be a problem. Today's networks are highly dynamic, since people use mobile devices, and very large in scale. Current infrastructure frameworks are not scaling with the potential number of service descriptions, cannot map the dynamics in the network, and have severe problems with the frequency and number of messages necessary to maintain such an infrastructure.

The thesis presented in this paper focuses on exactly this problem of conflicting goals in the mobile agent infrastructure, trying to map scalability (quantity) and flexibility (dynamics) into a unified solution framework – QuickLinkNet - that supports mobile agents in their autonomous behaviour. Thereby, timing bounds caused by using mobile devices in logical networks and scalability of globally distributed service descriptions are the major focus of attention. First an exploration of state-of-the-art systems is presented and drawbacks regarding the requirements of mobile agent systems on level 2 are uncovered. Starting from these requirements, the desired properties of the new solution framework are derived and then further specified. This leads to a software architecture that comprises of three components that form the framework: QuickLink, Service Juggler, and APPLICOOVER. The framework is then put into practice by implementing all QuickLinkNet components and by evaluating their functionality.

The QuickLinkNet components establish a transparent, logical two-tiered network which can be used as a service-oriented infrastructure for mobile agent systems of level 2 and for other distributed applications in highly dynamic networks as well. The two tiers of the logical network comply with the two views the entire network has to provide for the developer: a local view and a global view. Each view (and each tier) is realized by its own separate network structure and emphasizes different system properties and behavioural patterns. The local view, realized by the components QuickLink and ServiceJuggler, helps to control the high network dynamics in a local context and the necessity to provide up-to-date service descriptions. This is in principal achieved by a self-adapting network administration mechanism. The localized management of dynamics leads in turn to moderate timing bounds in the global view and enables the quantitative scalability of service descriptions, as realized by the component APLICOOVER on the basis of existing technologies.

Danksagung

Ich möchte mich an dieser Stelle bei allen bedanken, die zum Gelingen dieser Arbeit beigetragen haben.

An erster Stelle danke ich meinem Betreuer Prof. Dr. Wilhelm R. Rossak für die umfassende Unterstützung während der gesamten Promotionszeit. Auf seinen Rat und Beistand konnte ich mich auch bei schwierigen Fragen und in kritischen Situationen immer verlassen.

Ebenso danke ich meinem zweiten Betreuer Prof. Erich Stein für die konstruktiven Gespräche und seine beständige Unterstützung, mein Promotionsvorhaben umzusetzen.

Mein besonderer Dank gilt Frau Prof. Dr. Ilka Philippow für ihre Bereitschaft, als Gutachterin zu fungieren und für ihre zahlreichen hilfreichen Hinweise.

Weiterhin danke ich allen meinen Kollegen und Freunden am Lehrstuhl für Softwaretechnik für die kooperative Atmosphäre, die Forschung zu einer spannenden und angenehmen Tätigkeit macht. Insbesondere möchte ich Dr. Christian Erfurth, Dr. Andreas Hartmann, Torsten Dettborn, Christoph Henniger und Christian Schachtzabel für die vielen fruchtbaren Diskussionen und fachkundigen Kommentare danken. Bedanken möchte ich mich auch bei den ehemaligen Studenten Steffen Spranger, Thomas Hentrich und Karsten Hauser, die durch ihre Studien- und Diplomarbeiten zu dieser Arbeit beigetragen haben.

Ich danke auch allen Freunden, die immer die richtigen Worte und Wege zur Motivation fanden.

Ganz besonders herzlich danke ich meiner Familie für ihre ununterbrochene Unterstützung und insbesondere meinen Eltern, die mich stets in meinem Weg bestärkten. Und ich danke meiner geliebten Frau Elke, die in all den Jahren immer für mich da war und niemals am Erfolg meiner Arbeit zweifelte.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Das Internet und dessen Entwicklung aus Anwendersicht 1
- 1.2 Ein kurzer Blick auf Overlaynetzwerke und mobile Agentensysteme 4
- 1.3 Der Beitrag dieser Dissertation 6
- 1.4 Überblick über diese Arbeit 7

I Verteilte Systeme im Internet **9**

2 Offene Kommunikationssysteme **11**
- 2.1 Referenzmodelle für Schichtenmodelle 11
- 2.2 Internet - Schichtung und Funktionen 12
  - 2.2.1 Netzwerkschicht 12
  - 2.2.2 Internetschicht 15
  - 2.2.3 Transportschicht 18
  - 2.2.4 Anwendungsschicht 20
- 2.3 Zusammenfassung und Schlussfolgerungen 22

3 Konstruktionsparadigmen verteilter Systeme **25**
- 3.1 Begriffsbestimmung 25
- 3.2 Organisationsstrukturen verteilter Systeme 26
- 3.3 Konstruktionsziele von Middleware 28
  - 3.3.1 Transparenz 29
  - 3.3.2 Offenheit 30
  - 3.3.3 Skalierbarkeit 31
- 3.4 Zusammenfassung 35

4 Architekturmodelle verteilter Systeme **37**
- 4.1 Client-Server-Modell 37
- 4.2 Peer-to-Peer-Modell 39
  - 4.2.1 Aufbau und Eigenschaften 39
  - 4.2.2 Anwendungsebenen 40
  - 4.2.3 Klassifikation und architekturelle Parameter 42
  - 4.2.4 Konkrete Peer-to-Peer-Architekturen 43

4.2.5 Zusammenfassung Peer-to-Peer-Architekturen 56
4.3 Mobile Agentensysteme 57
4.3.1 Mobile Agentensysteme und mobile Agenten 58
4.3.2 Mobile Agentensysteme als verteilte Systeme 59
4.3.3 Eine mögliche Taxonomie für mobile Agentensysteme . . . 61
4.3.4 Anforderungen an Infrastrukturdienste für MAS der Stufe 2 62
4.3.5 Standards für Agentensysteme 63
4.3.6 Konkrete mobile Agentensysteme 71
4.3.7 Zusammenfassung mobile Agentensysteme 77

II QuickLinkNet - ein Infrastrukturdienst-Framework für MAS 79

5 Konzepte für Infrastrukturdienste für ein MAS der Stufe 2 81
5.1 Anforderungen und gewünschte Systemeigenschaften 81
5.2 Analyse und Strukturierung 82
5.2.1 Teilmenge 1 - globale Sicht 83
5.2.2 Teilmenge 2 - lokale Sicht 84
5.2.3 Schnittmenge der Teilmengen 1 und 2 85
5.3 Umsetzungsvorschlag 86
5.4 Thesen 87

6 Übersicht und allgemeine Beschreibung von QuickLinkNet 89
6.1 Architektur und Zusammenspiel der Komponenten 89
6.2 Spezielles Vokabular 91
6.3 Das logische Netzwerk 94
6.3.1 Aufbau des Netzwerkes 94
6.3.2 Zusammenwirken der Infrastrukturelemente 99
6.4 Zusammenfassung 101

7 QuickLink 103
7.1 Der Infrastrukturdienst QuickLink im Überblick 104
7.1.1 Aufgaben und Randbedingungen des logischen Netzwerkes 104
7.1.2 Vorbetrachtungen und Analyse zum logischen Netzwerk . . 105
7.1.3 Vorbetrachtungen und Analyse zur Leistungsmessung . . . 107
7.1.4 Aufbau und Architektur 110
7.2 Die Softwarekomponente QuickLink 111
7.3 Die Softwarekomponente CycleManager 112
7.3.1 Aufbau und Realisierung 112
7.3.2 Funktionsweise und Vorgänge 124
7.4 Die Softwarekomponente Kicker 129
7.4.1 Self-Healing 129

7.4.2 Die konkrete Aufgabe Kickers . . . 129
7.4.3 Funktionsweise . . . 130
7.5 Die Interfaces . . . 132
7.5.1 Priorities . . . 132
7.5.2 Performance . . . 133
7.5.3 Quicklink . . . 134
7.5.4 OwnServices . . . 134
7.5.5 PushedQuickLinkInfos . . . 135
7.6 Die Softwarekomponente PerformanceAnalyser . . . 136
7.6.1 Aufbau und Architektur . . . 136
7.6.2 Funktionsweise . . . 138
7.7 Zusammenfassung . . . 139

8 ServiceJuggler 141
8.1 Der Infrastrukturdienst ServiceJuggler im Überblick . . . 142
8.1.1 Analyse zu Anwendungsdienste verwalten . . . 142
8.1.2 Analyse zu Region verwalten und adaptieren . . . 144
8.1.3 Analyse zu lokale und globale Sicht verbinden . . . 146
8.1.4 Aufbau und Architektur . . . 146
8.2 Die Softwarekomponente ServiceManager . . . 148
8.2.1 Die Aufgaben des ServiceManagers . . . 148
8.2.2 Vorgänge und Funktionsweise . . . 148
8.3 Die Softwarekomponente ServiceRegistration . . . 154
8.3.1 Aufgaben und Funktionsweise . . . 154
8.3.2 Aufbau der Dienstbeschreibungen . . . 154
8.4 Die Softwarekomponente ServiceRegistry . . . 156
8.4.1 Der Dienstzugang . . . 157
8.4.2 Realisierung des Verzeichnisdienstes . . . 158
8.4.3 Dienstverwaltung und Kommunikation mit APLICOOVER 159
8.5 Die Interfaces . . . 162
8.6 Zusammenfassung . . . 163

9 APLICOOVER 165
9.1 Der Infrastrukturdienst APLICOOVER im Überblick . . . 166
9.1.1 Aufgaben und Randbedingungen des logischen Netzwerkes 166
9.1.2 P-Grid als geeignete Technologie . . . 167
9.1.3 Architektur des Infrastrukturdienstes APLICOOVER . . . 169
9.2 Die Softwarekomponente Core . . . 169
9.3 Die Softwarekomponente Storage . . . 171
9.3.1 Dienstbeschreibungen und Datentypen in APLICOOVER . 171
9.3.2 Die Hauptklasse StorageService . . . 171
9.3.3 Die Klasse SearchJob . . . 173

9.4 Die Softwarekomponente Statistics . . . 174
9.5 Die Softwarekomponente Messaging . . . 174
9.6 Die Interfaces . . . 175
9.6.1 RemoteAdministration . . . 175
9.6.2 RemoteServices . . . 176
9.7 Funktionsweise und Vorgänge APLICOOVERs . . . 177
9.7.1 Start, Bootstrapping und Herunterfahren . . . 177
9.7.2 Veröffentlichen, Aktualisieren und Löschen . . . 178
9.7.3 Suche nach Anwendungsdiensten . . . 179
9.7.4 Andere Suchmethoden in APLICOOVER . . . 181
9.7.5 Das Cacheprotokoll APLICOOVERs . . . 181
9.8 Zusammenfassung . . . 184

10 Zusammenfassung QuickLinkNet **185**

III Evaluierung QuickLinkNet **189**

11 QuickLink **191**
11.1 Randbedingungen der Vernetzung . . . 191
11.2 Netzwerkbelastung durch QuickLink . . . 192
11.2.1 Paketgrößen . . . 192
11.2.2 Paketaufkommen . . . 197
11.2.3 Resultierende Netzwerklast . . . 198
11.2.4 Diskussion Netzwerklast . . . 199
11.3 Praktische Evaluation des logischen Netzwerkes von QuickLink . . . 199
11.3.1 Messumgebungen . . . 200
11.3.2 Zuverlässigkeit . . . 203
11.3.3 Der Eintritt ins Netzwerk . . . 215
11.3.4 Rechenzeit . . . 221
11.4 Praktische Evaluation des PerformanceAnalysers . . . 230
11.4.1 Messumgebungen und Messmethodik . . . 231
11.4.2 Ergebnisse und Diskussion . . . 234
11.5 Zusammenfassung . . . 237

12 ServiceJuggler **239**
12.1 Messumgebungen . . . 239
12.2 Migration der Daten eines ServiceRegistry . . . 240
12.2.1 Messmethodik . . . 240
12.2.2 Ergebnisse Ethernet . . . 241
12.2.3 Ergebnisse Wireless LAN . . . 243
12.3 Anmeldezeiten am ServiceRegistry . . . 245

12.3.1 Messmethodik . . . 245
12.3.2 Ergebnisse Ethernet . . . 246
12.3.3 Ergebnisse Wireless LAN . . . 248
12.4 Zusammenfassung . . . 249

13 APLICOOVER **251**
13.1 Testumgebung . . . 251
13.2 Testmethodik . . . 252
13.3 Ergebnisse . . . 253
13.4 Zusammenfassung . . . 253

14 Evaluation der Thesen **255**

15 Zusammenfassung und Ausblick **257**

Literaturverzeichnis **261**

A Weitere Informationen zu QuickLinkNet **273**
A.1 Weitere Informationen . . . 273
A.2 Anhang - QuickLink . . . 273
A.2.1 Ermittlung der Leistungswerte . . . 273
A.2.2 Aufbereitung der Leistungswerte . . . 275
A.2.3 Umrechnung der Leistungswerte in Prioritätswerte . . . 276
A.2.4 Benchmark . . . 277
A.3 Anhang - ServiceJuggler . . . 279
A.4 Anhang - APLICOOVER . . . 280

Abbildungsverzeichnis

2.1 OSI- und Internet-Modell im Vergleich 13

3.1 Struktur eines verteilten Systems als Middleware 28

4.1 Interaktionsmuster und Rollenverteilung im Client-Server-Modell . . 38
4.2 Interaktionsmusters im Peer-to-Peer Modell 39
4.3 Architektur von und Kommunikation bei Napster 44
4.4 Anmeldevorgänge von Gnutella-Peers 45
4.5 Routing der Suchanfragen im Freenet-Netzwerk 49
4.6 Suchen im P-Grid 51
4.7 Aufbau des Chord Rings 53
4.8 Die Softwarearchitektur von JXTA 54
4.9 Interaktion im mobilen Agentensystem 58
4.10 Funktionsintegration in mobile Agentensysteme 60
4.11 Aufbau eines MASIF-konformen Agentensystems 66
4.12 MASIF-Infrastrukturelemente und deren Zusammenspiel 67
4.13 Die Infrastruktur von SOMA 72
4.14 Der Infrastrukturaufbau von TRACY 74

5.1 Isolierte Mengen von Eigenschaften 82
5.2 Mögliche Architektur eines Gesamtsystems 87

6.1 Die Architektur von QuickLinkNet 91
6.2 QuickLinkNets zwei logische Ebenen 94
6.3 Untere Netzwerkebene 96

7.1 Der Infrastrukturdienst QuickLink im Framework QuickLinkNet . . 103
7.2 Struktur der Softwarekomponenten von QuickLink 110
7.3 Die funktionale Struktur des CycleManagers 113
7.4 Der beispielhafte Datensatz einer QuickLink-List 115
7.5 Die beispielhafte Struktur einer QuickLink-List 116
7.6 Der beispielhafte Datensatz einer Priority-List 118
7.7 Die beispielhafte Struktur einer Priority-List 119
7.8 Der beispielhafte Aufbau einer Zyklusnachricht 120
7.9 Der beispielhafte Aufbau einer Aktualisierungsnachricht 120

7.10 Das Zustandsdiagramm des CycleManagers 124
7.11 Das Zustandsdiagramm der Softwarekomponente Kicker 130
7.12 Die funktionale Struktur des PerformanceAnalysers 137

8.1 Der Infrastrukturdienst ServiceJuggler im Framework QuickLinkNet 141
8.2 Die Architektur von ServiceJuggler 147
8.3 Das Zustandsdiagramm des ServiceManagers 149
8.4 Zustände eines Diensteintrages 160

9.1 APLICOOVER im Framework QuickLinkNet 165
9.2 APLICOOVERs Softwarekomponenten im Überblick 169
9.3 Das Cache-Protokoll APLICOOVERs 182

11.1 Der Messaufbau 1 im Ethernet 202
11.2 Der Messaufbau 2 im Wireless LAN 203
11.3 Die Zuverlässigkeitsmessungen im Ethernet 206
11.4 Zuverlässigkeitsmessungen ohne Begrenzung 208
11.5 Zuverlässigkeitsmessungen mit 3 Datenströmen ohne Begrenzung . . 209
11.6 Zuverlässigkeitsmessungen im Wireless LAN 210
11.7 Zeitreihen von Thinkpad im Wireless LAN 213
11.8 Zeitreihen der PING-Messreihe im Wireless LAN 214
11.9 Eintrittsmessungen im Ethernet 217
11.10 Eintrittsmessungen im Wireless LAN 219
11.11 Der Messaufbau 3 - Verarbeitungszeiten 222
11.12 Die Verarbeitungszeiten von Zyklusnachrichten 223
11.13 Die Verarbeitungszeiten von Updatenachrichten 226
11.14 Verarbeitungszeiten von Zyklusnachrichten im Vergleich 229
11.15 Verarbeitungszeiten von Updatenachrichten im Vergleich 231
11.16 Leistungskennzahlen der Messrechner im Vergleich 235

12.1 Messumgebungen zur Evaluation ServiceJugglers 240
12.2 Datenmigration zwischen zwei ServiceRegistry 240
12.3 Messergebnisse Datenmigration Ethernet 242
12.4 Messergebnisse Datenmigration WLAN 244
12.5 Schema Messung der Anmeldezeiten 246
12.6 Messergebnisse der Anmeldezeiten Ethernet 247
12.7 Messergebnisse der Anmeldezeiten WLAN 249

A.1 Das Klassendiagramm des Verzeichnisdienstes 279

Tabellenverzeichnis

8.1 Dienstbeschreibung nach FIPA XC00079 155
8.2 Antwort auf eine Suchanfrage nach FIPA XC00079 156

10.1 Rollen und nötige Komponenten in QuickLinkNet 186

11.1 Einige technische Daten der verwendeten Messrechner 201
11.2 Einige technische Daten der verwendeten Netzwerkgeräte 201
11.3 Messumgebung: Die Parameter QuickLinks 202
11.4 Die durchgeführten Messreihen im Ethernet 204
11.5 Die durchgeführten Messreihen im Wireless LAN 205
11.6 Zuverlässigkeitsmessungen im Ethernet 207
11.7 Zuverlässigkeitsmessungen im Wireless LAN 211
11.8 Eintrittsmessungen im Ethernet 218
11.9 Eintrittsmessungen im Wireless LAN 220
11.10 Die durchgeführten Messreihen zur Rechenzeit 222
11.11 Die Verarbeitungszeiten von Zyklusnachrichten 224
11.12 Die Verarbeitungszeiten von Updatenachrichten 227
11.13 Verarbeitungszeiten von Zyklusnachrichten im Vergleich 230
11.14 Verarbeitungszeiten von Updatenachrichten im Vergleich 232
11.15 Die Messrechner für die Leistungsmessungen 233
11.16 Ergebnisse Leistungskennzahlen im Vergleich 236

12.1 Messergebnisse Datenmigration Ethernet 241
12.2 Messergebnisse Datenmigration WLAN 243
12.3 Messergebnisse der Anmeldezeiten Ethernet 247
12.4 Messergebnisse der Anmeldezeiten WLAN 248

A.1 Die im Benchmark verwendeten Klassen und deren Methoden 278

1 Einleitung

1.1 Das Internet und dessen Entwicklung aus Anwendersicht

Vor etwa 20 Jahren war die Computerwelt noch durch wenige stationäre Arbeitsstationen mit festem Netzwerkanschluss geprägt. Als Anwender konnte man sich sicher sein, dass die Computer ständig liefen und zuverlässig erreichbar waren. Es gab selten Änderungen in den Netzwerkstrukturen, die Zusammenarbeit war meist auf kleine, abgetrennte, vertrauenswürdige Netzwerkbereiche und Maschinen beschränkt und die zusammenarbeitenden Computer waren sehr oft homogener Natur, so dass die wenigen damaligen *verteilten Systeme* meist eng miteinander gekoppelt waren. Eine sich ständig ändernde Verteilung von Computern, Informationen, Daten und Diensten im Netzwerk war nur schwer vorstellbar. Doch mit dem Durchbruch der Internettechnologie auf Grund seiner Killeranwendung *World Wide Web (WWW)* zu Beginn der neunziger Jahre des letzten Jahrhunderts hat sich die Welt der Computer und Computernetzwerke erheblich geändert. Die große Nachfrage privater und kommerzieller Nutzer nach dem neuen Informationsmedium WWW führte zu zwei Effekten: Zum einem gab es einen Boom bei den Netzwerkzugängen, welcher zum massiven Ausbau der Kernkommunikationsnetze seitens der Netzwerkbetreiber führte. Zum anderen stieg auch die absolute Anzahl an Computern enorm an. So kletterte die Zahl der über Domänennamen im DNS-System des Internets registrierten Computer von weniger als einer Million im Januar 1992 auf über 433 Millionen im Januar 2007 [Int07].

Zeitgleich zum Ausbau der Netzwerke entwickelte sich ein anderer Trend: Zu den festinstallierten Computern kamen mobile Computer wie Notebooks und Handheld-Geräte wie PDA sowie Mobiltelefone hinzu. Es folgten andere digitale Endgeräte wie Digitalkameras, digitale Musikspeicher und Navigationssysteme, wobei der aktuelle Trend in Richtung der funktionalen Verschmelzung solcher mobilen Endgeräte zu mobilen Multifunktionsgeräten geht. Mit der Entwicklung mobiler Endgeräte selbst ging auch die Entwicklung von Technologien zur Vernetzung dieser einher. So entstanden z.B. im Bereich der mobilen Computer die Netzwerktechnologien Wireless LAN und Bluetooth, im Bereich der Mobiltelefone GPS und UMTS. Auch wenn die drahtlosen, funkbasierten Zugangsnetze in Bezug auf Leistungsfähigkeit, Verbindungsqualität und Zuverlässigkeit bis heu-

te nicht mit den Festnetzen konkurrieren können, war damit die Grundlage zur Internetanbindung mobiler Endgeräte geschaffen. Zusammenfassend kann das Internet heute durch

- eine riesige Anzahl von vernetzten Endgeräten mit
- heterogenen Betriebssystemen,
- heterogenen Netzwerkanbindungen,
- höchst unterschiedlichen Ressourcen und
- einem Netzwerk, bestehend aus technologisch heterogenen Netzwerkabschnitten mit
- hoher Leistungsfähigkeit und Zuverlässigkeit in den Kernnetzwerken und
- niedriger und schwankender Leistungsfähigkeit und Zuverlässigkeit in den Peripherienetzwerken

charakterisiert werden, dessen alles verbindende Element das *Internetprotokoll (IP)* darstellt.

Als Anwender nutzen wir heute außer dem WWW ganz selbstverständlich auch verschiedenste andere Dienste und Anwendungen im Internet wie Email, Datentransferdienste, aber auch komplexere Internetdienstleistungen wie z.B. E-Commerce-Dienste. Solche Dienste werden uns an bestimmten, uns bekannten, i.d.R. festen Dienstzugangspunkten im Internet, den Dienstservern, offeriert. Mit Hilfe eines Browsers oder, bei komplexeren Diensten mit einer speziellen Clientsoftware, sprechen wir die Dienste über die festen Dienstzugangspunkte an. Und natürlich tun wir dies auch wie selbstverständlich von unseren mobilen Endgeräten aus, die heute fast überall einen Zugang zum Internet finden.

Das eben beschriebene Szenario folgt einem für Internetanwendungen typischen Interaktionsmuster, dem softwaretechnisch die *Client-Server-Architektur* zugrunde liegt. Typisch ist hierbei die feste Rollenverteilung in Dienstanbieter *(Server)* und Dienstnutzer *(Clients)*, die auch der bei dieser Architektur angenommenen Verteilung der physischen Ressourcen im Internet entspricht: Ein Server ist i.d.R. eine starke Rechenmaschine mit fester Kopplung an ein Festnetz, die Dienste anbietet, und ein Client ist ein schwächerer Computer mit ebenfalls schwächerer Anbindung an das Internet, der nur Dienste nutzt. Dieser Architekturstil gilt als allgemeintypisch für Internetanwendungen und die meisten Internetanwendungen sind heute noch nach dieser Architektur gebaut.

Die feste Verteilung der Rollen und Zugriffsmuster im Internet kann heute aber nicht mehr für alle Anwendungen als adäquat angesehen werden. Zum einen besitzen die heutigen Computer, zumindest die festangeschlossenen Arbeitsstationen, meist mehr Rechenleistung und Netzwerkbandbreite als sie für

die Bewältigung ihrer Arbeit brauchen. Dementsprechend niedrig ist ihre Auslastung. Zum anderen wollen viele Internetnutzer selbst Dienste und Informationen anbieten und gleichzeitig Dienste anderer Internetteilnehmer nutzen[1]. Daher entstehen seit Ende der neunziger Jahre zunehmend Internetanwendungen, die nach der *Peer-to-Peer-Architektur* geformt werden. Bei diesem Architekturstil bietet ein Computer Dienste an und nutzt auch Dienste anderer Computer. Die Rollenverteilung in Dienstnutzer und Diensterbringer ist dabei nicht fest verteilt, sondern kann wechseln oder beide Rollen können gleichzeitig angenommen werden. Jeder Computer wird daher bezüglich der Rollenverteilung als ein *Peer* (ein Ebenbürtiger) bezeichnet. Diese Bezeichung eines Computers im zugehörigen Interaktionsmuster mit gleichzeitiger gegenseitiger Dienstnutzung und -inanspruchnahme gab diesem Architekturstil auch seinen Namen. Unterstellt man einen hohen Verteilungsgrad von Ressourcen, so verspricht die Peer-to-Peer-Architektur bzgl. Rechenlast- und Netzwerklastverteilung gegenüber der Client-Server-Architektur erhebliche Vorteile. Moderne verteilte Anwendungen nutzen deshalb genau diese Architektur. Die bekanntesten Vertreter der Peer-to-Peer-Architektur sind wohl die allseits beliebten File-Sharing-Anwendungen wie z.B. Gnutella [Kan01] und Freenet [CSWH01]. Diese Peer-to-Peer-Anwendungen spannen ein virtuelles Netzwerk auf, in dem jeder Teilnehmer anderen Teilnehmern Daten zum Herunterladen anbieten und gleichzeitig selbst von anderen Teilnehmern angebotene Daten herunterladen kann. Ein anderer ganz aktueller Trend im Bereich der Geschäftsanwendungen sind die WebServices und die damit verbundene *dienstorientierte Architektur (Service-oriented architecture – SOA).* SOA-Anwendungen nutzen eine dienstorientierte *Middleware*, für die sich eine Peer-to-Peer-Architektur anbietet, um große verteilte Anwendungen zu bauen, da sie der natürlichen Verteilung der Dienste im Internet besser entspricht. Doch die Entwicklung des Internets zu einem riesigen, unüberschaubaren, dynamischen Informationsraum wird man auch mit reinen Peer-to-Peer-Architekturen nicht beherrschen können, sie werden lediglich einen Beitrag liefern können. Vor diesem Hintergrund sind *mobile Agenten*, eine besondere Form *mobilen Codes*, ein vielversprechender Ansatz, sich im Informationsraum Internet zurecht zu finden.

Hinter dem Ansatz von mobilem Code steht die Idee, z.B. Daten nicht vom Server durch das Netzwerk zum Client zu transportieren und dann erst auszuwerten, sondern den Code zur Auswertung der Daten zum Server zu schicken *(zu migrieren)*, dort ausführen zu lassen und dann nur die Ergebnisse zum Client zurückzusenden. Wenn die Größe des Codes kleiner als die zu durchsuchende Rohdatenmenge ist, bietet diese Methode Vorteile in der Netzwerkbelastung gegenüber dem Transport von Rohdaten. Ebenso kann, alternativ oder ergänzend,

[1] Die Veränderung der Organisationsform des WWW von der zentral orientierten Client-Server-Architektur hin zu verteilten Architekturen und dessen sich in diesem Zusammenhang für den Nutzer ändernde Wahrnehmbarkeit bildet einen zentralen Aspekt einer Entwicklung, die heute unter dem Begriff Web 2.0 [O'R05, Wik07e] zusammengefasst wird.

eine eventuell besonders rechenintensive Aufgabe vom Client zum Server ausgelagert werden, was bei leistungsschwachen Clientgeräten, wie sie mobile Endgeräte oft darstellen, vorteilhaft ist. Nach der Erledigung der Aufgabe wird der mobile Code aus dem Speicher des Servers gelöscht, da er nicht mehr gebraucht wird.

Mobile Agenten können als ein neues Programmierparadigma für verteilte Systeme und verteilte Anwendungen [Vig98] betrachtet werden. Sie sind kleine mobile Softwareentitäten, die einerseits die Möglichkeit zur Codemigration von mobilem Code und andererseits die Eigenschaften von Softwareagenten in sich vereinen. Softwareagenten sind autonome Prozesse, die sich durch folgende Eigenschaften [JW98] auszeichnen:

- Autonom (kann eigenständig reagieren),
- Reaktiv (reagiert rechtzeitig auf Änderungen in seiner Umgebung)
- Proaktiv (initiiert Aktionen, die seine Umgebung beeinflussen) und
- Kommunikativ (kann Informationen mit Benutzern und anderen Agenten austauschen).

Softwareagenten lassen sich erst durch die zusätzlich vom mobilen Code ererbte Fähigkeit zur Migration als mobile Agenten klassifizieren. Sie sind also im Gegensatz zu normalen Softwareagenten mobil. Im Gegensatz zu normalem mobilen Code besitzen mobile Agenten aber die erweiterten Fähigkeiten von Softwareagenten. Dies drückt sich vor allem im Verhalten mobiler Agenten aus. Während beispielsweise die Migration von mobilem Code von außen initiiert wird, entscheidet der autonome mobile Agent selbst, wann und wohin er migriert. Mobiler Code migriert nur einmal vom Client zum Server und wird dann von diesem terminiert, während ein mobiler Agent durchaus mehrfach auf verschiedene Server nacheinander migrieren kann, bevor er zu seinem Ausgangspunkt mit den Ergebnissen seiner Arbeit zurückkehrt und sich eventuell selbst terminiert.

Ist ein mobiler Agent ausreichend intelligent programmiert und sind seine Fähigkeiten bezüglich Autonomie, Reaktivität, Proaktivität und Kommunikativität entsprechend ausgeprägt, so stellt er eine ideale Lösung zur Navigation und Aufgabenlösung in unübersichtlichen Informationsräumen wie dem heutigen Internet dar. Allerdings muss ihm der Informationsraum Internet auch die von ihm benötigten Informationen zur Verfügung stellen, ihm also die passende Infrastruktur bieten.

1.2 Ein kurzer Blick auf Overlaynetzwerke und mobile Agentensysteme

In Folge der Forschung an mobilen Agentensystemen in den letzten Jahren existieren viele Referenzimplementierungen, von denen mittlerweile einige, wie

z.B. *TRACY2* von der *the agent factory GmbH* [Taf06] und *enago mobile* von der *IKV++ AG* [TA04], Produktstatus erreicht haben. Allerdings konzentrierte sich die Forschungsgemeinschaft viele Jahre nur auf die zentralen Themen mobiler Agentensysteme wie geeignete Programmiersprachen, Kommunikationsmuster zwischen Agenten, Standardisierung, Sicherheit, Fehlerbehandlung und Entwicklungstechniken mobiler Agenten, um nur einige zu nennen. Alle diese Arbeitsgebiete innerhalb der mobilen Agentensysteme fokussierten auf den mobilen Agenten selbst oder auf die Dienste der Agentenplattform, welche die Ausführungsumgebung mobiler Agenten darstellt.

Die intensive Forschungsarbeit im Bereich mobiler Agenten an der Friedrich-Schiller-Universität Jena bezog sich bisher vor allem auf Leistungsaspekte mobiler Agentensysteme [LSt07a, LSt07b] im Allgemeinen und die Anwendbarkeit mobiler Agentensysteme in den Domänen E-Business und Soziale Netzwerke [LSt07c] im Besonderen. Peter Braun optimierte mit seiner Softwarekomponente *Kalong* [Bra03] die Migration mobiler Agenten als einen der Kernleistungsparameter eines mobilen Agentensystems. Christian Erfurth verbesserte mit seinem Framework *ProNav* [Erf04] die Wahrnehmbarkeit der virtuellen Welt, die einen mobilen Agenten umgibt, und steigerte so die möglichen autonomen und proaktiven Eigenschaften mobiler Agenten bezüglich Routenplanung und Migration. Allerdings wurde in der Forschung bisher die Frage der Art und Weise der Vernetzung der einzelnen Agentenplattformen und damit auch die Frage nach der Verteilung von Dienstinformationen vernachlässigt. Dabei ist die Frage, ob und wann ein mobiler Agent eine benötigte Information bekommt, für die Leistungsfähigkeit eines mobilen Agentensystems essentiell wichtig.

Mobile Agentensysteme nutzen zur Vernetzung meist die Peer-to-Peer-Architektur. Ein reines Peer-to-Peer-Netzwerk bildet aber einen vollständig vermaschten Graphen, ist also strukturlos. Der Kommunikationsaufwand zum Verteilen von Dienstinformationen in einem solchen strukturlosen Netzwerk ist aufgrund von Skalierungsproblemen nur schwer beherrschbar. Daher muss das Peer-to-Peer-Netzwerk zumindest zur Verwaltung und Verteilung von Dienstinformationen strukturiert werden. Eine solche, dem Peer-to-Peer-Netzwerk "übergezogene" Struktur nennt man *Overlay-Struktur* und das so entstehende virtuelle Netzwerk *Overlaynetzwerk* [JAN02].

Es existieren in Forschung und Praxis nur wenige spezielle Ansätze zur Strukturierung mobiler Agentensysteme. Die wenigen existierenden Ansätze vernachlässigen die in der Praxis auftretenden, durch dynamisches Verhalten von Ressourcen verursachten Vernetzungsprobleme völlig oder gehen auf diese nur rudimentär ein. Mobile Endgeräte mit ihrem stark dynamischen Verhalten scheinen als Träger mobiler Agentenplattformen beim Entwurf dieser Systeme keine Rolle gespielt zu haben oder wurden bewusst ausgelassen. Auch die Frage der Mengenskalierung ist meist nicht gelöst oder der Lösungsansatz skaliert nur unter der

Annahme einer nicht vorhandenen bzw. geringen Netzwerkdynamik. Viele Entwürfe mobiler Agentensysteme verweisen lapidar auf eine dem Agentensystem unterliegende Middleware, ohne auf diese und ihre Eigenschaften genauer einzugehen. Doch gerade die Struktur des Overlaynetzwerkes und die daraus resultierenden Konsequenzen legen die Leistungsfähigkeit des gesamten Systems und damit auch die des mobilen Agentensystems fest.

1.3 Der Beitrag dieser Dissertation

In dieser Dissertation werden mobile Agentensysteme aus dem Blickwinkel der Verteilung, speziell der Verteilung von Diensten und Dienstinformationen, untersucht. Dabei liegt das besondere Interesse auf:

- der Verteilungsstruktur mit Fokus auf seine beiden leistungsbezogenen Parameter, nämlich
- Mengenskalierbarkeit unter den Bedingungen der Endgerätezahl des heutigen Internets in Verbindung mit
- der potentiellen Netzwerkdynamik, wie sie von mobilen Endgeräten hervorgerufen wird.

Die im vorherigen Abschnitt aufgeworfenen Fragen der Verteilung, der Mengenskalierbarkeit und der Netzwerkdynamik sind aber keine nur auf mobile Agentensysteme bezogene Probleme, sondern treten bei allen stark verteilten Anwendungen gleichermaßen auf. Deshalb können neue Ansätze zur Lösung der Probleme auf alle verteilten Anwendungen, die unter ähnlichen Randbedingungen wie mobile Agentensysteme arbeiten und ähnliche Aufgaben zu erfüllen haben, angewendet werden. So werden in dieser Arbeit die auftretenden Probleme anhand der Anforderungen mobiler Agentensysteme untersucht, ohne dass die gefundenen Lösungen der Allgemeingültigkeit entbehren.

Zur Verdeutlichung der Probleme werden Leistungsbedingungen[2] abgeleitet, die auf Grundlage von Szenarien mobiler Agentensysteme der *Stufe 2* [Erf04] aufgestellt wurden. Mobile Agentensysteme der Stufe 2 bieten zusätzlich zu den bisherigen mobilen Agentensystemen der Stufe 1 Dienste zur autonomen Migration und Wegfindung für mobile Agenten an. Mit diesen Diensten unterstützen sie mobile Agenten in der Wahrnehmung ihrer Umwelt, ermöglichen ihnen die autonome Wahl ihrer nächsten Zielpunkte, zu denen sie migrieren wollen und bieten die Möglichkeit zur Optimierung der Routenplanung. Alle diese Dienste beruhen auf den zur Zeit der Dienstinanspruchnahme auf der Agentenplattform

[2] Die Leistungsbedingungen und Anforderungen werden im Kapitel 4.3.4 detailliert erläutert und diskutiert.

vorhandenen Informationen, welche ausreichend aktuell sein sollten. Die beteiligten mobilen Agenten müssen jeweils ausreichend intelligent programmiert und in der Lage sein, diese Dienste auch sinnvoll nutzen zu können. Ferner sollte ihre zu lösende Aufgabe eine komplexe Reiseroute, zumindest aber mehrere anzulaufende Zielpunkte, erfordern.

Der konkrete Beitrag dieser Arbeit besteht in:

- der Analyse der beteiligten wissenschaftlichen Gebiete und ihrer Einflüsse auf die Leistungsparameter verteilter Systeme,
- der Herausarbeitung der architekturellen Nachteile bestehender mobiler Agentensysteme und anderer artverwandter Peer-to-Peer-Systeme,
- der Herausarbeitung der Anforderungen mobiler Agenten und mobiler Agentensysteme der Stufe 2 an ihre grundsätzlichen Infrastrukturdienste,
- der Vorstellung eines neuen, mehrschichtigen Infrastrukturdienst-Frameworks *QuickLinkNet*, welches die Anforderungen mobiler Agentensysteme der Stufe 2 erfüllt und in
- der Evaluation des neuen Infrastrukturdienst-Frameworks und dem Nachweis seiner Leistungsfähigkeit.

1.4 Überblick über diese Arbeit

Die Arbeit ist in drei Teile gegliedert.

Im ersten Teil werden die Konzepte und Architekuren der beteiligten Themengebiete Kommunikationssysteme (Kapitel 2) und verteilte Systeme (Kapitel 3) vorgestellt und die Auswirkungen auf die Leistungsparameter von Middleware aufgezeigt und diskutiert. Kapitel 4 widmet sich den Architekturmodellen verteilter Systeme, wie Client-Server, Peer-to-Peer und mobile Agentensysteme. Der State-of-the-Art von Overlaystrukturen wird anhand konkreter Peer-to-Peer-Systeme und konkreter mobiler Agentensysteme dargelegt, mit den Anforderungen eines MAS der Stufe 2 verglichen und die Defizite werden aufgezeigt.

Im zweiten Teil der Dissertation werden in Kapitel 5 anhand der Anforderungen eines MAS der Stufe 2 die gewünschten Systemeigenschaften abgeleitet, ein Lösungsvorschlag unterbreitet und die Thesen der Dissertation aufgestellt. In den sich anschließenden Kapiteln 6 bis 10 wird die Architektur und Funktionsweise des neuen Middlewareframeworks *QuickLinkNet* und seiner Komponenten vorgestellt und detailliert beschrieben.

Schlussendlich werden im dritten Teil die Komponenten des neuen Frameworks evaluiert und die erwartete Leistungsfähigkeit nachgewiesen.

I

Verteilte Systeme im Internet

2 Offene Kommunikationssysteme

Dieses Kapitel führt den Leser in den Bereich der Architektur von Kommunikationssystemen ein, ohne aber einen Gesamtüberblick über dieses große Gebiet zu geben. Vielmehr werden die für diese Dissertation notwendigen, grundlegenden Begriffe und Zusammenhänge der *Schichtenarchitektur*, des im Bereich der Kommunikationssysteme vorherrschenden Architekturmodells, erläutert. Im Speziellen wird dann auf die Grundlagen und den Aufbau des Internetschichtenmodells und den Internetprotokollstapel als technischer Grundlage des Internets eingegangen. Es werden die Auswirkungen der Konstruktionseigenschaften des Internetprotokollstapels bezüglich der geographisch globalen Ausdehnung des Internets, der Skalierung und der Netzwerkdynamik diskutiert.

2.1 Referenzmodelle für Schichtenmodelle

1977 wurde das OSI-Referenzmodell (*Open Systems Interconnection*) als Standard für offene Kommunikationssysteme von der ISO *(International Organization for Standardization)* [ISO05] verabschiedet. Obwohl es bisher nie vollständig geschlossen implementiert wurde [HPS00], hat es sich bis heute als wichtigstes Referenzmodell gehalten.

Das OSI-Modell ist in sieben funktionale Schichten gegliedert, welche jeweils aufeinander aufbauen. Der Gesamtfunktionsumfang des OSI-Modells deckt den gesamten Bereich von der elektrischen Signalisierung und elektrisch/mechanischen Schnittstellenbeschreibung auf Schicht 1 bis zu den Kommunikationsanwendungen, welche komplexe Kommunikationsprotokolle darstellen, auf Schicht 7 ab. Jede Schicht erfüllt gewisse Funktionalitäten, welche sie in Form von genormten Schnittstellen *(Diensten)* der jeweils darüber gelegenen Schicht zur Nutzung anbietet. Aus der Sicht der unterliegenden Schicht werden ihre Funktionalitäten in ihr eingeschlossen, sie gehorcht dem Prinzip der *Kapselung*.

Eine Schicht n nimmt immer nur die Dienste der jeweils direkt darunter liegenden Schicht *n-1* in Anspruch. Dabei wird der von einem Dienst gebotene Funktionsumfang immer im Zusammenspiel aller darunterliegenden Schichten geschaffen. Da Schicht n zur Erstellung ihrer Dienste aufgrund des *Kapselungsprinzips* aber nur direkt auf Dienste der Schicht *n-1* zurückgreift, nimmt sie die Funktionsumfänge der darunterliegenden Schichten nicht direkt wahr, sie sind für sie *transparent*.

Die Prinzipien der *Kapselung* und *Transparenz* gelten für alle Schichtenmodelle. Sie erleichtern dem Programmierer das Umgehen mit komplexen Sachverhalten und ermöglichen die Abstraktion von technischen Details. Allerdings führt die zunehmende Abstraktion auch zum Verlust eventuell nützlicher Informationen.

Ende der sechziger Jahre des vorigen Jahrhunderts begann, vom US-amerikanischen Department of Defence *(DoD)* initiiert, die Entwicklung des gleichnamigen DoD-Modells mit dem Ziel, Großrechner weiträumig vernetzen zu können. Die erste Implementierung nannte sich *ARPANET* und ging 1969 online. Im Laufe seiner Weiterentwicklung entstanden 1973 die Protokolle *Internet Protocol (IP)* und *Transmission Control Protocol (TCP)*, mit welchen auch die Vernetzung von heterogenen Netzwerken möglich wurde. 1983 wurde das ARPANET in das rein militärisch genutzte MILNET und das nunmehr zivile ARPANET getrennt, wobei TCP und IP die Standardprotokolle in beiden Netzen wurden. Damit war der Grundstein für das heutige Internet gelegt. Aufgrund der Bedeutung und Dominanz dieser beiden Protokolle in ihren Schichten spricht man heute auch vom *TCP/IP-Protokollstapel*.

Basierend auf dem DoD-Modell bildet das Internet-Modell den gleichen Funktionsumfang wie das OSI-Modell ab, allerdings auf nur vier Schichten verteilt. In Abbildung 2.1 sind die Funktionsumfänge der Schichten des OSI-Referenzmodells und des Internetmodells mit einigen Protokollbeispielen gegenübergestellt.

2.2 Internet - Schichtung und Funktionen

2.2.1 Netzwerkschicht

Die Netzwerkschicht[1] des Internet-Modells deckt einen großen technologischen Bereich ab, welcher u.a. die Netzwerkhardware mit ihren mechanischen und elektrischen Schnittstellen, die Bitübertragung, Fehlersicherung auf Bitebene, Flussregelung und Medien umfasst. Die in dieser Schicht beschriebenen Funktionalitäten sind sehr vielfältig und zum Großteil technologieabhängig. Daher ist eine Kommunikation zwischen heterogenen Technologien auf dieser Schicht nicht möglich und auch nicht vorgesehen. Im Betriebssystem eines Rechners entspricht diese Schicht einer Kommunikationshardware samt ihrer Treibersoftware. Als Beispiel sei hier eine Ethernetnetzwerkkarte genannt, die nur mit anderen Ethernetadaptern der gleichen Technologiestufe kommunizieren kann. Es ist natürlich möglich, mit solch einer Technologie ein eventuell auch größeres Netzwerk

[1] Diese Schicht wird im deutschen Sprachraum auch als *Netzzugangsschicht* oder *Sicherungsschicht* bezeichnet. Andererseits werden die auf dieser Schicht verwendeten Protokolle als *Netzwerkprotokolle* bezeichnet. Wegen der Möglichkeit zur eindeutigen sprachlichen Abgrenzung gegenüber der Internetschicht wird in dieser Dissertation die unterste Schicht des Internetmodells als *Netzwerkschicht* bezeichnet.

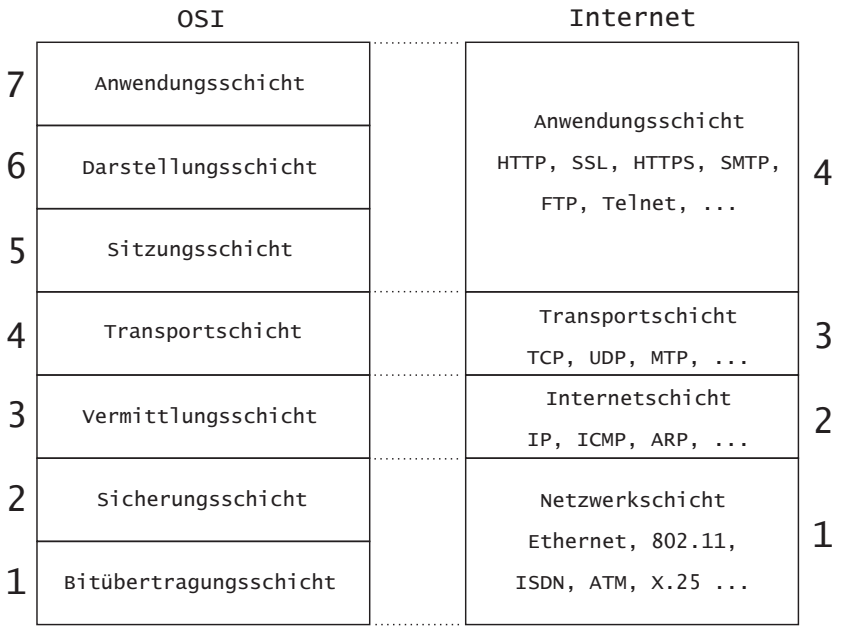

Abbildung 2.1: Funktionale Schichtenteilung des OSI-Modells und des Internet-Modells im Vergleich

aufzubauen, wie große lokale Netzwerke beweisen. Sobald man aber an die geographische Technologiegrenze stößt, kann keine weitere Kommunikation stattfinden. Natürlich kann man durch geeignete Netzwerkgeräte wie Brücken auch technologisch ungleiche Netzwerke verbinden und das lokale Netzwerk erweitern, aber ein gut funktionierendes globales Netzwerk wird man damit nicht realisieren können, da die verwendeten Algorithmen und Funktionalitäten globale Bedingungen (Latenzzeiten, Fehlerraten, Adressenvorrat und Mengenskalierung) nicht berücksichtigen.

Die Netzwerkschicht ermöglicht aus der Gesamtsicht auf das Internetmodell den grundlegenden Netzzugang und eine einigermaßen zuverlässige Kommunikation auf einer technologiegebundenen, kurzen Teilstrecke im Internet. Die (relative) Hardwarenähe der Software sorgt für eine gute Reaktionsgeschwindigkeit auf dieser Schicht, was vor allem bei drahtlosen Kommunikationsnetzwerken mit ihren inhärent hohen Fehlerraten und dynamischen Effekten, wie Feldstärkeschwankungen, von Vorteil ist. Der Eintritt in ein und Austritt aus einem Funknetzwerk, wie er z.B. bei Mobilität von Endgeräten auftritt, wird schon auf dieser Ebene weitgehend automatisch behandelt und abgefangen, führt aber trotzdem zu spürbaren Verzögerungen und kann bis zum Abbruch der Verbindung auf höheren Schichten führen, da diese wegen des Transparenzprinzips (siehe dazu auch Abschnitt 2.1) über die Gründe für Verzögerungen nicht informiert werden.

Die meisten lokalen Netzwerktechnologien nutzen konzeptionell ein gemeinsames, geteiltes Medium. Dies resultiert entweder aus ihrer Netzwerktopologie oder, wie bei Funknetzwerken, aus den physikalischen Gegebenheiten der Luftschnittstelle. Gesendete Nachrichten breiten sich auf einem geteilten Medium allseitig aus und erreichen so alle angeschlossenen Netzwerkteilnehmer nacheinander. Eine Nachricht muss also das gesamte Netzwerk durchdringen, sie gehorcht dem *Diffusionsprinzip*. Diese Art der Nachrichtenausbreitung, bei der eine Nachricht immer alle Teilnehmer erreicht, nennt man auch *Broadcasting (Rundfunk)*. Sie tritt in dieser Form z.B. beim klassischen Ethernet mit Bustopologie und beim WLAN im Ad-hoc-Modus auf. Ein Teilnehmer prüft erst nach dem korrekten Empfang der Nachricht anhand der Empfängeradresse, ob die Nachricht für ihn bestimmt war und verwirft sie, wenn dies nicht der Fall ist. Aus dem Diffusionsprinzip und den Broadcasteigenschaften bei Netzwerken mit geteiltem Medium ergeben sich daher zwei interessante Eigenschaften: Zum einen darf immer nur eine Nachricht zur selben Zeit unterwegs sein, d.h. nur ein Teilnehmer darf senden und alle anderen Teilnehmer müssen empfangen. Dies muss durch die Zugriffssteuerung der Netzwerkprotokolle auf dieser Schicht sichergestellt werden. Zum anderen wird jede Nachricht automatisch immer an alle Empfänger zugestellt, da einzig die Broadcastkommunikation [Ste04] verwendet wird. Dieser Fakt ist sehr nützlich, wenn man ohnehin Nachrichten an alle Empfänger versenden will, da auf der Netzwerkschicht kein zusätzlicher Kommunikationsaufwand entsteht. Der Kommunikationsaufwand bei n teilnehmenden Stationen im Netzwerk beträgt lediglich konstant eine Nachricht.

Bei anderen Netzwerktopologien wie der ringförmigen beim Token-Ring-Netzwerk oder bei sternförmiger Topologie, wie beim geswitchtem Ethernet, besteht das Netzwerk aus einzelnen Punkt-zu-Punkt-Verbindungen. Beim Token-Ring-Netzwerk ist jeder Rechner mit zwei Nachbarrechnern durch einzelne, unabhängige Punkt-zu-Punkt-Verbindungen verbunden. Nachrichten werden nur in Richtung eines Nachbars weitergegeben und von diesen und dessen Nachbarn solange weitergereicht, bis sie den adressierten Empfänger erreicht haben. Beim Broadcast wird die Nachricht einmal im Kreis geleitet, bis sie wieder beim Sender ankommt. Der Kommunikationsaufwand beim Broadcast mit n Stationen im Netzwerk liegt bei $n-1$ Nachrichten. Im Vergleich zum Kommunikationsaufwand einer Nachricht für nur einen Empfänger ist dies aber nicht schlimm, da dieser unter den gleichen Bedingungen zwischen 1 Nachricht im Best-Case und $n-2$ Nachrichten im Worst-Case liegt. Ferner können die gerade nicht benutzten Teilstrecken des Token-Ring-Netzwerkes für andere Nachrichten verwendet werden, so dass im Gegensatz zum Diffusionsnetzwerk immer mehrere Nachrichten gleichzeitig unterwegs sein können. Token-Ring-Netzwerke sind heute auf Grund ihrer hohen Hardwarekosten nur noch wenig verbreitet und spielen im Alltag praktisch keine Rolle mehr.

Das geswitchte Ethernet besteht ebenfalls aus exklusiv belegten Punkt-zu-Punkt-Verbindungen, die aber zwischen den teilnehmenden Arbeitstationen und einem zentralen Verarbeitungsgerät, dem Switch, in sternförmiger Struktur angeordnet sind. Der Switch empfängt alle Nachrichten aller Teilnehmer und leitet sie entsprechend der Empfängeradresse dem Empfänger exklusiv zu. Bei Broadcast leitet der Switch die Nachricht allen Teilnehmern gleichzeitig zu. Der Kommunikationsaufwand beträgt daher bei Einzelsendungen wie bei Broadcastnachrichten immer konstant 2 Nachrichten. Prinzipiell den gleichen Aufbau und das gleiche Verhalten hat ein Wireless LAN im Infrastrukturmodus, bei dem die Aufgabe des Switches vom Access-Point übernommen wird. Ein Unterschied zum geswitchtem Ethernet besteht im konkurrierenden Medienzugriff der Netzwerkteilnehmer beim Funknetzwerk. Dieser hat allgemein auf das zeitliche Leistungsverhalten von Funknetzwerken negative Auswirkungen, wirkt sich aber beim Vergleich des Kommunikationsaufwandes von Einzelsendungen gegenüber Broadcastnachrichten nicht aus.

2.2.2 Internetschicht

Die Aufgabe der Internetschicht[2] umfasst aus technischer Sicht die *globale Verbindung* von Rechnern über technologisch heterogene Teilnetzwerke der Netzwerkschicht hinweg. Dabei decken sich Netzwerke der Internetschicht nicht zwingend mit den Netzwerken der Netzwerkschicht, auch wenn aus organisatorisch-administrativen und technischen Gründen oft eine Überdeckung vorherrscht. So ist es möglich, in einem IP-Netzwerk mehrere Netzwerke der Netzwerkschicht zusammenzufassen oder auch ein Netzwerk der Netzwerkschicht in mehrere IP-Netzwerke aufzuteilen.

Aus logischer Sicht erfüllt die Internetschicht eine Abstraktion vom unterliegenden Netzwerk insoweit, dass sie eine *Ende-zu-Ende*-Sichtweise präsentiert und das unterliegende Netzwerk transparent macht. Zu diesem Zweck werden auf der Schicht des Internetprotokolls *global gültige, logische IP-Adressen* eingeführt, welche ein *IP-Netzwerk und einen IP-fähigen Rechner zusammen identifizieren.* Ein Sender schickt ein Paket mit Hilfe der Empfängeradresse direkt an einen Empfänger (daher Ende-zu-Ende-Sicht), ohne sich weiter um den Transport und das Ankommen des Paketes zu kümmern. Das Vermitteln des Paketes übernehmen zusätzliche Netzwerkgeräte, sogenannte *Router*, welche das Paket auf Basis des netzwerkidentifizierenden Teils der Empfängeradresse in Richtung Zielnetzwerk zum nächsten Router weiterleiten, bis das Zielnetzwerk erreicht ist. Der

[2] Diese Schicht des Internetmodells wird im deutschen Sprachraum meist als *Vermittlungs-* oder *Netzwerkschicht* bezeichnet. Diese Bezeichnung stammt ursprüngich aus der OSI-Termininologie. Im englischsprachigen Raum wird diese Schicht dagegen oft als *Internetschicht* bezeichnet. Weil das Internetprotokoll (IP) [Pos81a] auf dieser Schicht konkurrenzlos ist, wird in dieser Dissertation die nach Auffassung des Autors treffendere Bezeichnung verwendet.

letzte Router, der das Zielnetzwerk mit dem Internet verbindet, liefert das Paket dann an den Empfänger (auf Netzwerkschicht) aus. Das Internetprotokoll wird auch als *verbindungslos* bezeichnet, weil jedes gesendete Paket beim Weiterleiten durch die Router einzeln für sich behandelt wird. Die Router sorgen u.a. auch für eine gewisse Überlaststeuerung im Netzwerk, da verstopfte Leitungen berücksichtigt werden und Pakete in einem solchen Fall über andere Teilnetze und damit über alternative Routen geschickt werden können. Daher ist der Weg eines Pakets durch das Internet nicht garantiert, sondern nur, dass ein Paket irgendwie zum Empfänger geleitet wird, falls es nicht verloren geht. Aus diesen Charakteristika des Internetprotokolls resultieren unzuverlässige Verbindungseigenschaften, da weder die richtige Reihenfolge noch die Vollständigkeit aller übertragenen Pakete beim Empfänger garantiert werden kann.

Die abstrakte Ende-zu-Ende-Sichtweise der Internetschicht bildet die Grundlage für den recht einfachen Umgang mit den Netzwerken aus Programmierersicht: Der Programmierer verteilter Systeme kann einen Zielrechner ansprechen, solange dessen IP-Adresse bekannt ist. Über die Vernetzung, Topologien und hardwarenahe Protokolleigenschaften muss er sich keine Gedanken mehr machen. Andererseits geht mit der Ende-zu-Ende-Sichtweise ein Verlust an Informationen über die Netzwerkstruktur einher: Dadurch entsteht ein völlig unstrukturiertes Netzwerk auf der Internetschicht. Dieser Verlust an Strukturinformationen wird in den kommenden Abschnitten und im nächsten Kapitel noch eine große Rolle spielen.

Im letzten Abschnitt hat sich das Broadcasting als effektive Kommunikationsmethode zur Nachrichtenübertragung auf der Netzwerkschicht herausgestellt. Auch das Internetprotokoll unterstützt Broadcasting über spezielle Broadcastadressen, welche in jedem IP-Netzwerk existieren. Es ist immer die numerisch höchste Adresse in einem IP-Netzwerkbereich. Sendet man ein Paket an genau diese Adresse, wird es an alle am Netzwerk angeschlossenen Rechner weitergeleitet. Die Umsetzung des Broadcasts erfolgt in jedem IP-Netzwerk wieder durch den Broadcastmechanismus des jeweils unterliegenden Netzwerkprotokolls.

Da das IP-Protokoll jedoch global adressiert, unterstützt es automatisch auch Broadcasts in andere IP-Netzwerke hinein. Dazu wird einfach die Broadcastadresse des Netzes, in welchem man den Broadcast auslösen will und dessen Broadcastadresse man sich selbst bilden kann, verwendet. Im Extremfall könnte man auch einen Broadcast in das gesamte Internet auslösen. Diese Funktionalität birgt starke Sicherheitsrisiken bezüglich Angriffe auf Netzwerke durch Überschwemmung mit Broadcastnachrichten in sich. Das offensichtlichste Bedrohungsszenario wäre wohl ein *Denial-of-Service* Angriff, bei dem durch Überlastung, hier durch Überlastung des Netzwerkes, Dienste oder ganze Netzwerke arbeitsunfähig gemacht würden. Um sich dieser Bedrohung zu entziehen, werden Broadcasts in andere IP-Netzwerke von jedem Router standardmäßig blockiert,

während lokale Broadcasts im eigenen IP-Adressenbereich zugelassen werden. Neben der relativ geringen Netzwerkbelastung bei der Verwendung von Broadcasts sollte man beachten, dass immer alle von der Nachricht erreichten Computer die Nachrichten auch verarbeiten müssen. Daher sollte man Broadcast nur dort einsetzen, wo auch alle Computer erreicht werden sollen.

Die heute immer noch meistverwendete Version des Internetprotokolls ist die Version 4 (*IPv4*) und stammt aus dem Jahr 1981 [Pos81a]. Aus der 32-Bit langen Adresse ergibt sich eine theoretische Anzahl von knapp 4,3 Milliarden Adressen, was in der damaligen Zeit als mehr als ausreichend angesehen wurde. Zieht man aber die vielen Sonderadressbereiche und die Adressen mit fester Bedeutung (u.a. die Broadcastadressen) ab, reduziert sich die Zahl der verfügbaren Adressen erheblich. In der heutigen Zeit, wo mit Paradigmen wie *All over IP* [Tri03] die gesamte Kommunikationsinfrastruktur auf das Internetprotokoll umgebaut wird, ist dies natürlich nicht mehr ausreichend. Die Mengenskalierung der potentiellen Rechner im Internet ist mit IPv4 demnach nicht sichergestellt. Daher wird seit 1995 am Nachfolgeprotokoll der Version 6 (*IPv6*) gebaut. IPv6 bietet u.a. eine Adresslänge von 128 Bit [DH98] und damit einen Vorrat von $3{,}4 \times 10^{38}$ Adressen, womit die Mengenskalierbarkeit des Internets bezüglich der Adressenanzahl in der nächsten Zeit gelöst sein dürfte.

Die schon erwähnte Teilung einer IP-Adresse in einen netzwerkidentifizierenden und einen rechneridentifizierenden Teil führen zu einer Ortsbezogenheit einer IP-Adresse. Durch die Ortsbezogenheit der IP-Adresse ergeben sich Schwierigkeiten beim Auftreten von Endgerätemobilität, im speziellen beim Übertritt von mobilen Endgeräten in ein anderes IP-Netzwerk. Da die IP-Adresse nur das Endgerät im Zusammenhang mit einem bestimmten IP-Netzwerk identifiziert, ist die alte IP-Adresse beim Eintritt in das neue IP-Netzwerk nicht mehr gültig. Das Endgerät benötigt hier also eine neue IP-Adresse und deren Vergabe und Propagation an potentielle Kommunikationspartner benötigt Zeit. Im Allgemeinen wirkt sich Mobilität auf der Netzwerkschicht und auf der Internetschicht so aus, dass der Empfänger nicht mehr erreicht werden kann oder eine schon vorhandene Kommunikationsverbindung abbricht. Eine Lösung für diese Probleme stellen Mobile IPv4 [Per96, Per02] und die neue IP-Version IPv6 [JPA04] dar, in welche u.a. die Mobilitätsumfänge von Mobile IP mit integriert sind. Beide Versionen arbeiten nach dem Stellvertreterprinzip, wodurch neue Probleme wie Dreiecksrouting und Tunnelketten entstehen. Da IPv6 als Weiterentwicklung von IPv4 und Mobile IP jedoch Automatismen zur selbständigen IP-Adressenbildung integriert hat, viele beigestellte IP-Adressen *(co-located Care-of-Adressen)* pro Rechner ermöglicht und das Dreiecksrouting nur zur erneuten Kontaktaufnahme mit dem Stellvertreter nach dem Netzwerkwechsel verwendet, fallen die durch die Mobilität hervorgerufenen Probleme nicht ganz so stark ins Gewicht.

Eine Ortung und ein Ansprechen eines mobilen Endgerätes nach einem Netz-

werkwechsel ist daher zwar möglich, aber immer mit einer nicht unerheblichen Verzögerung verbunden. Dies drückt sich in geminderter Leistungsfähigkeit der Kommunikationsverbindung aus, die sich z.B. durch zusätzliche Latenzen und geringere Bandbreite auf Internetschicht äußern und sich zu den auf der Netzwerkschicht entstehenden Verzögerungen addieren. Zusammenfassend lässt sich feststellen, dass Mobilität auf der Internetschicht möglich, aber mit deutlichen Leistungseinbußen, zumindest bei Netzübergang, zu rechnen ist. Damit ist der abfangbaren Netzwerkdynamik gerade auf den höheren Schichten eine Grenze gesetzt, die sich stark nach der auf der Netzwerkschicht eingesetzten Technologie richtet. Mobilität kann auf Internetebene wegen des Transparenzprinzips nicht direkt erkannt werden, sondern drückt sich nur indirekt in Leistungsverlust aus, welcher natürlich auch andere Ursachen haben kann. Eine zuverlässige Erkennung von Mobilität wäre nur durch einen schichtenübergreifenden Informationsfluss von der Netzwerkschicht nach oben möglich.

2.2.3 Transportschicht

Die Transportschicht stellt das Bindeglied zwischen den kommunikations- und netzwerkbezogenen, unteren Schichten des Internetprotokollstapels und der Anwendungsschicht dar. Wie im letzten Abschnitt beschrieben, leistet das IP-Protokoll die globale Erreichbarkeit von Rechnern im Internet, ohne das Ankommen der gesendeten Pakete zu garantieren (unzuverlässige Verbindungseigenschaften). Die Aufgabe der auf der Transportschicht angesiedelten Protokolle bestehen daher vor allem in der Herstellung einer gewissen Dienstgüte, die das IP-Protokoll nicht leistet, und der Sicherstellung der Kommunikation zwischen einzelnen Prozessen auf den Rechnern. Die Zuordnung von Paketen zu einzelnen Prozessen erfolgt nach dem *Port-Prinzip*. Jedem kommunikationswilligen Prozess wird ein bestimmter Port (eine Portnummer) entweder fest oder dynamisch zugeteilt. Gesendete Pakete werden mit dieser Portnummer versehen und ankommende Pakete können anhand dieser Nummer einem Prozess zugeordnet werden. Erst dadurch wird eine echte Ende-zu-Ende-Verbindung und ein Nachrichtenaustausch zwischen Prozessen auf verschiedenen Rechnern möglich.

Es existiert heute eine Vielzahl verschiedener Transportprotokolle mit unterschiedlichen Verbindungsqualitäten und -eigenschaften, um den von den Anwendungsprotokollen auf der Anwendungsschicht geforderten unterschiedlichen Dienstgüten und -eigenschaften gerecht zu werden. An dieser Stelle soll aber nur auf die zwei wichtigsten, grundlegenden Protokolle *User Datagram Protocol (UDP)* [Pos80] und *Transmission Control Protocol (TCP)* [Pos81b] kurz eingegangen werden.

UDP ist ein Transportprotokoll, welches außer dem Port-Prinzip nicht viel mehr als die vom Internetprotokoll zur Verfügung gestellten Eigenschaften auf der Transportschicht bietet. Es ist daher unzuverlässig in Bezug auf Verlust, Vervielfältigung und Reihenfolgeänderung der transportierten Pakete. Allerdings erkennt UDP optional verfälschte Pakete und verwirft diese, so dass eine zuverlässige Fehlererkennung ermöglicht wird. Da UDP nicht verbindungsorientiert arbeitet, sind (auf Kosten der Zuverlässigkeit) höhere Datendurchsätze als bei TCP möglich. Die Übertragung von Daten mittels UDP kann Vorteile haben, wenn:

- nur wenige Daten übertragen werden sollen (z.B. nur ein Paket),
- die Daten nur eine kurze zeitliche Aktualität besitzen und sich eine wiederholte Übertragung im Fehlerfall wegen Veraltung der Daten nicht lohnt,
- die Effizienz des Datendurchsatzes oder eine geringe Netzwerkbelastung besonders wichtig ist oder
- von einer niedrigeren Schicht eine sehr zuverlässige Verbindung geboten werden kann.

UDP wird u.a. zum Streamen von Multimediadatenströmen bei Multicastanwendungen, von vielen Verwaltungswerkzeugen für lokale Netzwerke und zur Broadcastkommunikation auf Ebene der Transportschicht verwendet. Eine effiziente Broadcastkommunikation ist aber nur im lokalen IP-Bereich möglich, da Broadcasts in andere Netzwerke in der unterliegenden Internetschicht durch die Router, wie im vorigen Abschnitt beschrieben, verhindert werden.

TCP ist das Standardprotokoll im Internet und bietet einen zuverlässigen, bidirektionalen und verbindungsorientierten Datenaustausch zwischen zwei Prozessen. Zuverlässig bedeutet in diesem Zusammenhang, dass Pakete vollständig und in der richtigen Reihenfolge an den Zielprozess weitergeleitet werden, nachdem eventuelle Duplikate verworfen und fehlende Pakete wiederholt angefordert und übertragen wurden. Bevor Daten übertragen werden, baut TCP eine Verbindung zum Zielrechner auf und handelt mit diesem die Bedingungen für die Datenübertragung aus. Die gesendeten Daten müssen vom Empfänger bestätigt werden, es werden also immer Daten in beiden Richtungen übertragen (bidirektional). Die bidirektionale Verbindung ermöglicht auch einen gleichzeitigen Nutzdatenaustausch in beiden Richtungen (Vollduplex). Um den Empfänger der Daten nicht zu überlasten, bedient sich TCP einer ausgeklügelten Flusssteuerung. Ferner sind im TCP Mechanismen zur Stauerkennung und Maßnahmen zur Reaktion darauf implementiert, wobei verschiedene Implementierungen [Bra89, KHM00]

unterschiedliche Charakteristika aufweisen können. Der ansehliche Aufwand für diesen Komfort kostet allerdings im Vergleich zu UDP Leistung in der Nutzdatenrate, so dass TCP vor allem immer dann eingesetzt wird, wenn es auf:

- eine absolut zuverlässige Datenübertragung,
- ein großes Datenvolumen,
- eine große Übertragungsdistanz ankommt oder
- die Daten über eine inhärent unzuverlässige Netzwerkschicht übertragen werden müssen.

Zumindest eine dieser Bedingungen trifft auf die meisten Internetanwendungen zu, worin sich die Bedeutung und Anwendungsbreite von TCP begründet.

Aus Programmierersicht ermöglichen es die Protokolle der Transportschicht, Prozesse gezielt mit anderen Prozessen auf beliebigen Rechnern im Internet kommunizieren zu lassen, ohne sich um die Details des Datenaustausches über die zugrundeliegenden Basisnetzwerke kümmern zu müssen. Die mit der Mobilität von Endgeräten verbundenen Effekte, an denen schon die Internetschicht leidet, lassen sich auf der Transportschicht noch schlechter managen als auf der Internetschicht, da ausfallende Pakete bei UDP überhaupt nicht bemerkt werden und bei TCP die Mechanismen der Flusssteuerung und der Stauerkennung einsetzen, welche für Mobilität überhaupt nicht geeignet erscheinen und sich in einem extremen Leistungsverlust [FZ01] widerspiegeln. Mit speziell verbesserten TCP-Implementierungen wie in [FZ01] oder TCP Vegas [KHM00] können die leistungsschmälernden Effekte gemildert werden. Auch hier würde ein schichtenübergreifender Informationsfluss von der Netzwerkschicht in die Internetschicht und von dort in die Transportschicht die Reaktionsmöglichkeiten von verteilten Anwendungen verbessern können, obwohl dies dem Transparenzprinzip der Referenzmodelle widerspricht.

2.2.4 Anwendungsschicht

Auf der Anwendungsschicht[3] existiert eine Vielzahl unterschiedlicher Anwendungsschichtprotokolle, wie in Abbildung 2.1 angedeutet, welche die Dienste UDP und/oder TCP der Transportschicht nutzen. Diese Protokolle stellen komplexere und speziellere Kommunikationsdienste für Softwareanwendungen zur Verfügung, welche mit anderen Rechnern über das Netzwerk kommunizieren wollen.

[3] Der Name kommt von den Anwendungsprotokollen, welche auf dieser Schicht angesiedelt sind. Die Anwendungen selbst befinden sich außerhalb des Protokollstapels und nutzen die Dienste der Anwendungsschicht.

Deshalb sind diese Protokolle heutzutage i.d.R. in das Betriebssystem des Rechners integriert und qualifizieren es damit zu einem Netzwerkbetriebssystem (vgl. dazu auch Abschnitt 3.2). Netzwerkbetriebssysteme bieten natürlich auch die grundsätzlichen Kommunikationsprotokolle der Transportschicht als Dienste in Form von *Sockets* an.

Wichtige Anwendungsschichtprotokolle für verteilte Systeme sind, außer den grundsätzlichen Kommunikationssockets von TCP und UDP, z.B. die Protokolle (oder Dienste):

- *Dynamic Host Configuration Protocol (DHCP)* [Dro97], welches der automatischen IP-Adressenvergabe in IP-Netzwerken dient,
- *Domain Name Service (DNS)* [Moc87a, Moc87b] zur Auflösung logischer Rechnernamen in IP-Adressen,
- *Remote Procedure Call (RPC)* [Sri95] für den entfernten Prozeduraufruf,
- *Network File System (NFS)* [SCRT03] zum entfernten Dateizugriff,
- entfernte Methodenaufrufe für objektbasierte Systeme über die *Common Object Request Architecture (CORBA)* [The05] und über
- *JAVA Remote Method Invokation (JAVA RMI)* [Jav07a],
- *Secure Socket Layer (SSL)* [FKK96] bzw. dessen Weiterentwicklung *Transport Layer Security (TLS)* [DA99], ein verschlüsseltes TCP und
- Verzeichnisdienste wie das verbreitete *Lightweight Directory Access Protocol*
  (LDAP) [WHK97] und
- *Directory Enabled Networks (DEN)* [Str99].

Mit Hilfe dieser und anderer Dienste ist es möglich, komplexe Kommunikationsstrukturen aufzubauen, welche zur Realisierung verteilter Systeme gebraucht werden. Verteilte Systeme sind einerseits selbst Anwendungen, befinden sich also oberhalb der Anwendungs(protokoll)schicht und nutzen deren Protokolle, andererseits bieten sie noch mächtigere, Transparenz (siehe Abschnitt 3.3.1) schaffende Kommunikationsdienste für andere Anwendungen an, so dass sie auch als Frameworks von Anwendungsschichtprotokollen angesehen werden können.

2.3 Zusammenfassung und Schlussfolgerungen

Das allgemeine Problem der Mengenskalierung der am Internet teilnehmenden Rechner ist seit der Vorstellung von IPv6 konzeptionell gelöst. In Zukunft werden die Netzwerke sukzessive auf das neue Protokoll umgestellt. Die Umstellung wird durch die Tunnelungsmöglichkeiten von IPv4 Paketen im IPv6 Netzwerk wesentlich erleichtert.

Die Schichtenarchitektur des Internets vereinfacht aus Sicht des Programmierers den Umgang mit dem Netzwerk und reduziert den Aufwand für das Ansprechen entfernter Rechner. Der Programmierer bedient sich dabei der Protokolle der Anwendungsschicht des Internetprotokollstapels, welche ihm einfache und bequeme Kommunikationsdienste zur Verfügung stellen. Durch die zunehmende Abstraktion des Internetprotokollstapels nach oben hin nehmen die Transparenz, aber auch der Informationsverlust zu. Dies führt ab der Internetschicht mit seiner *Ende-zu-Ende-Sichtweise* zu einem unstrukturierten Netzwerk. Die Informationen über die natürliche Strukturierung der Netzwerke sind dadurch nicht mehr vorhanden und können folglich auch nicht genutzt werden. Dies zwingt verteilte Anwendungen dazu, zumindest im globalen Kontext Vorkehrungen zur Strukturierung und Verteilung von Dienstinformationen außerhalb des Internetprotokollstapels zu treffen oder diese Aufgabe einer speziellen Softwareschicht, der bei uns in Abschnitt 3.2 eingeführten *Middleware*, zu überlassen.

Die Einführung einer *Ende-zu-Ende-Verbindung* mit dem Transportprotokoll TCP ermöglicht es erstmals, sinnvolle Qualitätsparameter für eine Internetverbindung festzulegen und wahrzunehmen. Diese können sich aber eben nur auf die gesamte Verbindung und nicht auf einzelne Teilstrecken beziehen. Das Internetprotokoll stellt auch keine standardisierten Schnittstellen zur Übertragung von Qualitätsdaten *(Quality of Service - QoS)* von der Netzwerkschicht zur Transportschicht bereit, denn schichtenübergreifende Kommunikation war aber beim Entwurf des Internetprotokollstapels keine Konstruktionsanforderung, weil sie dem Transparenzprinzip des Schichtenmodells widerspricht. Der durch das Transparenzprinzip hervorgerufene Informationsverlust verhindert die Auswertung von Daten aus der Netzwerk- und der Internetschicht, was zumindest im lokalen IP-Netzwerkbereich mit wenigen oder nur einer unterliegenden Netzwerkteilstrecke Vorteile bieten könnte. Leistungswirksame Einflüsse auf die Ende-zu-Ende-Verbindung von TCP wie z.B. durch Router verursachte alternative Routen oder Endgerätemobilität können daher nicht ihrer Ursache nach detektiert werden, sondern machen sich nur indirekt durch Leistungsverlust bemerkbar.

Möchte man viele Rechner gleichzeitig mit denselben Informationen versorgen und dabei das Netzwerk nur wenig belasten, bietet sich die Broadcastkommunikation als probates Mittel dazu an. Broadcastkommunikation ist über alle Schichten des Internetprotokollstapels effektiv und effizient möglich. Auf der In-

ternetschicht wird die Broadcastkommunikation jedoch durch die Router auf den lokalen IP-Netzwerkbereich begrenzt, so dass Broadcast in der Praxis nur in diesem Bereich eingesetzt werden kann. Da Broadcast auf dem unzuverlässigen UDP basiert, müssen verteilte Anwendungen oder Middleware eventuell Vorkehrungen zur Fehlerbehandlung bei Datenverlust treffen. Die Vorteile effizienter Broadcastkommunikation zieht Nachteile in der Rechenbelastung der Computer nach sich, da alle Computer die Broadcastnachrichten verarbeiten müssen. Die könnte zumindest bei mobilen Endgeräten möglicherweise die Rechenkapazität spürbar belasten und Energievorräte vorzeitig verbrauchen.

3 Konstruktionsparadigmen verteilter Systeme

Mit der rasanten technologischen Entwicklung der Mikrocomputer in den achtziger Jahren des vergangenen Jahrhunderts und dem damit einhergehenden Preisverfall von Computerhardware wurde der Mikrocomputer erstmals ein wahres Massenprodukt. Damit entstand auch die Nachfrage nach der Vernetzung der plötzlich ansteigenden Menge von Computern in Industrie und im privaten Bereich. Diese Nachfrage konnte durch die sich ebenfalls rasant entwickelnde Technik der lokalen Netzwerke *(Local Area Network - LAN)* und dem rapiden Ausbau der Weitverkehrsnetze *(Wide Area Network - WAN)* in dieser Zeit erfüllt werden.

Der weitere Ausbau der Netzwerktechnologien in den neunziger Jahren des zwanzigsten Jahrhunderts ermöglichte dann die Ausbreitung und den Boom des Internets und seiner Killeranwendung World Wide Web *(WWW)*, des wahrscheinlich größten verteilten Systems weltweit. So ist es heute auch problemlos möglich, Computer mit unterschiedlicher Hard- und Software *(heterogene Computer)* miteinander zu vernetzen und weltumspannende Anwendungsprogramme, wie z.B. die beliebten File-Sharing-Tools zeigen, zu verwirklichen. Solche Systeme, die Ressourcen, wie z.B. Dienste, Daten, Rechenleistung oder Speicherplatz, auf verschiedenen Computern bereithalten und allen am System beteiligten Computern zur Nutzung bereitstellen, nennt man verteilte Systeme.

3.1 Begriffsbestimmung

Es existieren unterschiedliche Definitionen verteilter Systeme, welche nicht unbedingt miteinander übereinstimmen. Allerdings gibt es eine Schnittmenge von Merkmalen, welche in verschiedenen Definitionen [CDK02, RP99, TS03] vorkommen: Ein verteiltes System

- besteht aus einer Menge voneinander unabhängiger, vernetzter Computer,
- welche miteinander kommunizieren und
- dem Benutzer wie ein einzelnes, kohärentes System erscheinen.

Aus diesen Eigenschaften ergeben sich zwei wichtige Punkte zum Design verteilter Systeme. Zum einen müssen voneinander unabhängige Computer ausfallen

können, ohne dass die Funktion des Gesamtsystems wesentlich davon beeinflusst werden darf. Dies funktioniert aber nur, wenn die Rollenverteilung der Computer im System wechseln kann und sie vor dem Benutzer verborgen bleibt. Ebenfalls steht in diesem Zusammenhang die einfache Erweiterbarkeit eines verteilten Systems um neue Ressourcen und damit die Frage nach der Mengenskalierbarkeit, also der Frage, wie sich ein verteiltes System bezüglich bestimmter Leistungsparameter beim Hinzufügen sehr großer Mengen weiterer Ressourcen verhält.

Zum anderen erfordert die Illusion eines einzigen kohärenten Systems für den Benutzer das Verbergen der Kommunikation und der inneren Organisation und Struktur des Systems, eine sehr harte und in ihren Auswirkungen weitreichende Forderung. Gerade bei großflächigen und heterogenen verteilten Systemen stellt dies eine Herausforderung dar, da je nach Systemdynamik keine aktuelle, konsistente und für alle Computer und Nutzer gültige Sicht auf das Gesamtsystem existiert.

Diese Aufgaben werden in modernen verteilten Systemen von einer logischen Softwareschicht gelöst, die als *Middleware* bezeichnet wird. Die Middleware ist neben *verteilten Betriebssystemen* und *Netzwerkbetriebssystemen* nur ein Konzept zur softwaretechnischen Realisierung eines verteilten Systems.

3.2 Organisationsstrukturen verteilter Systeme

Aus Sicht der Software existieren drei klassische Ansätze zur Realisierung verteilter Systeme:

- Verteilte Betriebssysteme,
- Netzwerkbetriebssysteme und
- Middleware.

Verteilte Betriebssysteme bilden auf das Hardwarekonzept homogener Multicomputer ab: Mehrere homogene Prozessoren oder Computer verwenden z.B. einen gemeinsamen Speicher (streng gekoppeltes System) über einen gemeinsamen Speicherbus. Das verteilte Betriebssystem versucht dabei, eine globale Sicht auf alle eingebundenen Speicherkomponenten zu liefern und ihn wie einen einzelnen großen Speicher darzustellen. Dadurch wird eine sehr hohe Zugriffstransparenz erreicht. Allerdings ist das verteilte Betriebssystem wenig offen konstruiert: Durch die enge Kopplung homogener Computer sind verteilte Betriebssysteme als Konzept für verteilte Systeme im Internet nicht geeignet, da das Internet aus heterogenen Systemen besteht.

Netzwerkbetriebssysteme verwalten heterogene Computerhardware. Die Computer sind deshalb lockerer, z.B. über ein lokales Netzwerk, gekoppelt als bei einem verteilten Betriebssystem. Da die heterogene Hardware nicht mehr direkt über gemeinsame Speicher zusammenarbeiten kann, bieten die Computer ihre Ressourcen mit Hilfe primitiver Dienste (z.B. ssh, scp, ftp) an. Die Dienstzugangspunkte werden über ihr Netzwerkbetriebssystem (z.B. Linux, Windows ab Version 3.11) verwaltet. Netzwerkbetriebssysteme sind konzeptionell wesentlich einfacher aufgebaut als verteilte Betriebssysteme. Während verteilte Betriebssysteme versuchen, eine vollständige Verteilungstransparenz herzustellen, gelingt dies den Netzwerkbetriebssystemen nur zum Teil. Dafür sind Netzwerkbetriebssysteme robuster gegenüber Ausfällen einzelner Computer im System, da keine strenge Kopplung von Hardwareressourcen erfolgt. Früher entworfene verteilte Anwendungen verwendeten meist direkt die Programmierschnittstellen des verteilten Betriebssystems, wie die Kommunikationssockets der Transportschicht und die lokalen Dateisystemdienste. Dadurch konnte aber weder eine Verteilungstransparenz noch ein betriebssystemunabhängiger Zugriff auf die Ressourcen erreicht werden. Deshalb erscheint die Funktionalität und Leistungsfähigkeit von Netzwerkbetriebssystemen allein für die Konstruktion großer verteilter Systeme als nicht ausreichend geeignet.

Middleware ist das modernste Konzept für verteilte Systeme. Es versucht, die Vorteile der beiden vorherigen Konzepte zu kombinieren, indem es die Skalierbarkeit und Offenheit von Netzwerkbetriebssystemen mit der Transparenz und der damit verbundenen leichteren Nutzbarkeit verteilter Betriebssysteme verbindet. Eine Middleware stellt eine zusätzliche, verteilte Softwareschicht dar, welche die Ressourcen der im Netzwerk verteilten Computer verwaltet und in transparenter Weise ansprechbar macht. Middleware nutzen die primitiven Kommunikationsdienste von Netzwerkbetriebssystemen, um ihre komplexeren Dienste zu ermöglichen. Insofern sind sie von den Netzwerkfähigkeiten der Netzwerkbetriebssysteme direkt abhängig. Man kann sie deshalb auch als Erweiterung oder Aufsatz auf Netzwerkbetriebssysteme betrachten. Fast alle Betriebssysteme sind aber heute netzwerkfähig, so dass man die Netzwerkfähigkeit eines Betriebssystems heute praktisch nicht mehr als besonderes Klassifizierungsmerkmal ansieht, sondern diese stillschweigend voraussetzt. Ebenso wird die Unterstützung des Internetprotokollstapels heute als integraler Bestandteil aller gängigen, universellen Betriebssysteme angesehen. Vor diesem Hintergrund wird bei der Verwendung des Begriffs Middleware im weiteren Verlauf dieser Dissertation immer das Vorhandensein eines Netzwerkbetriebssystems vorausgesetzt und die Middleware in Bezug zu ihrem Einsatz im Internet betrachtet.

3.3 Konstruktionsziele von Middleware

Die Middleware wird, wie in Abbildung 3.1 dargestellt, als eine zusätzliche Verwaltungsschicht zwischen Netzwerkbetriebssystem und verteilter Anwendung eingefügt. Sie spricht lokale Ressourcen über Dienste des Betriebssystems an und stellt sie in Form von eigenen, betriebssystemunabhängigen Diensten über Standardschnittstellen verteilten Anwendungen zur Verfügung.

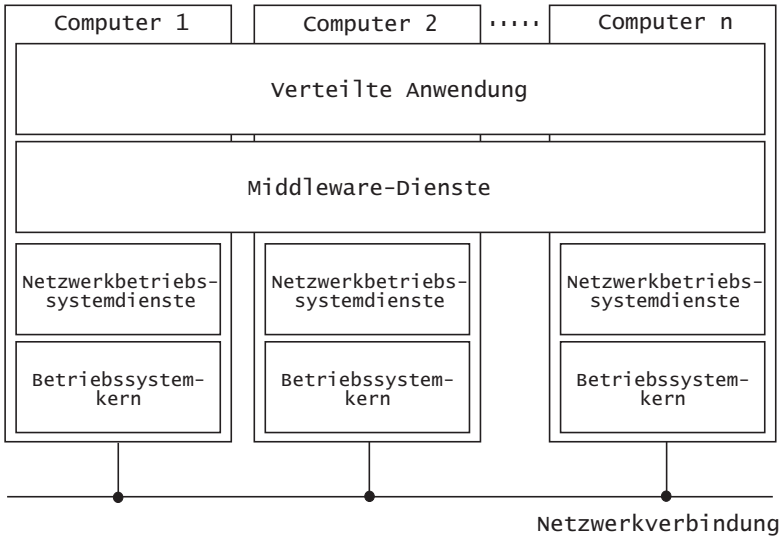

Abbildung 3.1: Struktur eines verteilten Systems als Middleware (angelehnt an [TS03], S. 55).

Das wichtigste Ziel ist dabei die Verdeckung der Hardware- und Softwareheterogenität der Plattformen. Die Ressourcen können somit in *transparenter* Weise angesprochen werden und machen die Dienste der Middleware *offen* für die Nutzung und das *Hinzufügen (Skalierbarkeit)* heterogener Plattformen. Anwendungen, die auf Basis einer Middleware implementiert werden, sollten daher nur Middleware-Schnittstellen nutzen und diese lokal nicht umgehen. Dazu muss die Middleware aber auch eine hinreichende Menge an Diensten zur Verfügung stellen, um die Funktionalität einer verteilten Anwendung nicht einzuschränken.

Die bei dieser Betrachtung aufgetauchten Eigenschaften *Transparenz*, *Offenheit* und *Skalierbarkeit* sollen nun in den nächsten Abschnitten als Konstruktionsziele von Middleware tiefergehend betrachtet werden.

3.3.1 Transparenz

Der Transparenzbegriff beschreibt das Verdecken bestimmter Vorgänge vor dem Benutzer und der Anwendung. Nach der Art der verdeckten Vorgänge innerhalb der Middleware können nahezu beliebig verschiedene Transparenzarten abgeleitet werden. Die folgende Aufzählung und Erklärung der wichtigsten Transparenzarten ist daher nicht abschließend, d.h. der Transparenzbegriff kann noch auf weitere Vorgänge und Eigenschaften ausgeweitet werden. Eine genauere Übersicht kann man [TS03] S. 21 ff. entnehmen.

Ortstransparenz: Middleware sollen Anwendungen mit Ressourcen verbinden, ohne dass ihre physischen Positionen, also ihre Standorte im System, erkennbar sind. Diese Art der Transparenz bezeichnet man auch als Ortstransparenz. Durch Ortstransparenz wird die tatsächlich vorhandene Verteilung der Ressourcen verdeckt. Um Ortstransparenz zu erreichen, werden Ressourcen mit einem logischen Namen versehen. Aus diesem geht aber der Standort einer Ressource nicht insgeheim, weil in z.B. verschlüsselter Weise im Namen beinhaltet, hervor. Über den logischen Namen kann die Ressource vom Benutzer oder der Anwendung angesprochen werden, wobei sich die Middleware um die Namensauflösung, die Adressenzuordnung und das Ansprechen der Ressource kümmert.

Verteilungstransparenz: Benutzer einer Anwendung sollen den zeitlichen Unterschied des Zugriffs auf nahe oder weit entfernte Ressourcen nicht spüren.

Zugriffstransparenz: Der Zugriff auf eine von einer Middleware verwalteten Ressource soll immer auf die gleiche Art und Weise erfolgen. Dabei ist es egal, ob sich die Ressource auf dem lokalen Computer oder im Netzwerk befindet.

Skalierungstransparenz: Das Hinzufügen von Ressourcen in und das Herausnehmen aus dem Gesamtsystem soll so vonstatten gehen, dass in die Middleware eingebundene Anwendungen dies möglichst weder bemerken noch davon beeinflusst werden.

Migrationstransparenz: Das Verschieben von Ressourcen (z.B. Diensten) soll so vonstatten gehen, dass in die Middleware eingebundene Anwendungen, oder zumindest die Benutzer der Anwendungen, dies möglichst nicht bemerken oder davon beeinflusst werden.

Fehler- und Ausfalltransparenz: Fällt eine Ressource oder eine Netzwerkverbindung aus, sollte die Anwendung weiterarbeiten können. Ein Leistungsabfall gilt hierbei als zumutbar.

Sprachtransparenz: Die Verwendung der Dienste und der Zugriff auf Ressourcen sollte unabhängig von der verwendeten Programmiersprache vonstatten gehen können.

Möglichst alle Transparenzanforderungen sollen von einer Middleware erfüllt werden. Da viele Transparenzarten aber Abhängigkeiten untereinander aufweisen, werden bei der Konstruktion von Middleware in der Praxis die Transparenzarten betont, die für das geplante System die besten Gesamteigenschaften versprechen. Betrachtet man z.B. eine verteilte Anwendung, die gleichartige Ressourcen in intelligenter Weise so in Anspruch nehmen soll, dass immer die leistungsstärkste verwendet wird, so werden sich Orts- und Verteilungstransparenz störend auf die Anwendung auswirken. Oder möchte ein mobiler Agent seine Route im Netzwerk optimal planen, so interessieren ihn die Verteilung und die Verbindungseigenschaften zu den Zielpunkten seiner Reise essentiell. Eine universelle und gleichzeitig für alle denkbaren Anwendungen optimale Middleware ist daher nicht realisierbar, da es immer kontroverse Zielvorgaben für die Konstruktion und die daraus folgenden Eigenschaften geben wird.

3.3.2 Offenheit

Ein weiteres Ziel von Middleware ist Offenheit. Ein offenes verteiltes System bietet Dienste mit Schnittstellen an, deren Syntax und Semantik nach Standardregeln *(Spezifikationen)* beschrieben ist. Alle Prozesse, die diese Schnittstelle benötigen und diese Spezifikation unterstützen, können mit dem Dienst kommunizieren. Durch die spezifizierte Schnittstelle ist die Funktion von der konkreten Implementierung des Dienstes getrennt. Erst dadurch wird Interoperabilität zwischen heterogenen Computern und Softwareplattformen und die Portabilität von Anwendungen möglich.

Verteilte Anwendungen werden heute oft betriebssystemunabhängig implementiert, indem sie gegen eine Middleware programmiert werden. Man verlässt sich auf die Implementierungen der Middleware, die einen großen Bereich heterogener Hardware unterstützen. Natürlich sind solche Anwendungen stark von der Middleware bzw. dem unterstützten Middlewarestandard abhängig. Offene Middleware sind durch die Vollständigkeit ihrer Schnittstellen und unterstützten Protokolle spezifiziert. Unvollständige Spezifikationen führen zu Abweichungen in der Implementierung und resultieren in Inkompatibilitäten zwischen Middleware gleichen Standards. Daraus resultieren natürlich auch Inkompatibilitäten von (eventuell gleichen!) Anwendungen und Anpassungsarbeiten bei der Portierung von Software auf andere Computersysteme. Aus diesen Gründen sind Middleware in der Praxis oft weniger offen als behauptet wird [TS03].

3.3.3 Skalierbarkeit

Die Eigenschaft Skalierbarkeit beschreibt die Erweiterbarkeit einer Middleware oder eines verteilten Systems, ohne durch diese wesentliche Verluste bei der Leistungsfähigkeit und eine wesentliche Erhöhung der administrativen Komplexität hinnehmen zu müssen. Neuman [Neu94] unterscheidet drei Komponenten der Skalierbarkeit:

- Die Anzahl der Benutzer oder Ressourcen eines verteilten Systems (Mengenskalierbarkeit),
- der Abstand zwischen den am weitesten auseinander liegenden Knoten (geographische Skalierbarkeit) und
- die Anzahl der administrativ unabhängigen Bereiche des verteilten Systems (Administrationsskalierbarkeit).

Da die Größe und Ausdehnung einer Middleware bei ihrem Entwurf nicht immer fest stehen, sollten immer Maßnahmen ergriffen werden, um ihm weitere Ressourcen einfach und problemlos auch in großen Mengen hinzufügen zu können. Die Ergebnisse dieser Maßnahmen formen die Eigenschaften der Skalierbarkeit einer Middleware und damit deren Leistungsverhalten im besonderen Maße. Deshalb wird in dieser Dissertation ein besonderes Augenmerk auf die Skalierbarkeit und Skalierungsprobleme gelegt. Die einzelnen Skalierungsarten werden in den folgenden Abschnitten näher erläutert.

3.3.3.1 Mengenskalierbarkeit

Bezüglich der Anzahl der unterstützten Ressourcen stellen vor allem *zentrale Konzepte* wie zentrale Dienste, zentrale Daten und zentrale Algorithmen Hemmnisse dar, die schon rein konzeptionell zu Engpässen und *Single-Point-of-Failure* Konstellationen führen.

Zentrale Dienste sind manchmal aus Sicherheits- und Administrationsgründen erwünscht, nur führt eine immer höhere Anzahl von Anfragen an einen zentralen Server zwangsläufig an dessen Leistungsgrenze: Die Antwortzeiten steigen immer weiter an, bis dass der Server überlastet ist und nicht mehr antworten kann. Damit ist der Dienst ausgefallen.

Eine ähnliche Auswirkung hat die zentrale Datenhaltung: Die zentrale Datenhaltung in einer Datenbank einer global erreichbaren Anwendung wäre rein mengenmäßig schon schwer beherrschbar, aber Millionen von gleichzeitigen Clientanfragen an einen einzelnen Server würden die zum Server hingehenden Kommunikationsnetzwerke hoffnungslos überlasten.

Zentrale Algorithmen, z.B. zur Optimierung des Routings in einem globalen Netzwerk, verursachen ähnliche Probleme: Das Sammeln einer vollständigen Information über alle Netzwerkverbindungen würde eine extrem hohe Netzwerklast erzeugen und Teile des Netzwerkes überlasten, so dass der positive Effekt des optimierten Routings ad absurdum geführt werden würde. Ein weiteres Problem stellt die Aktualität der Daten dar, denn bei hoher Netzwerkdynamik in einem globalen Netzwerk wären Informationen aus entfernten Netzwerkteilen bereits veraltet, bevor sie mit Hilfe des zentralen Algorithmus verarbeitet und die Ergebnisse wieder verteilt wären.

Aber auch dezentrale Ansätze können ihre Tücken haben: Wie schon in Abschnitt 2.2.3 festgestellt wurde, entsteht auf der Transportschicht mit seiner virtuellen Ende-zu-Ende-Verbindung ein unstrukturiertes (vollständig vermaschtes) virtuelles Netz, welches $n * (n - 1)$ gerichtete Verbindungen enthält. Stellt man sich eine verteilte Anwendung mit einem rein dezentralen Verteilungsansatz wie bei der idealen Peer-to-Peer-Architektur vor, in welcher jeder Peer seine Dienstinformationen an jeden anderen Peer senden will, so skaliert die Anzahl der zu versendenden Nachrichten nicht mit den im globalen Kontext vielleicht Millionen von beteiligten Computern.

3.3.3.2 Geographische Skalierbarkeit

Geographische Skalierbarkeit kann durchaus unabhängig von der Mengenskalierbarkeit betrachtet werden, obwohl ein mengenmäßig großes System meist auch geographisch weit verteilt ist. Verbindungen über geographisch große Entfernungen sind typischerweise durch eine große Anzahl von Teilstreckenverbindungen gekennzeichnet, welche auf der Netzwerkschicht durch viele verschiedene Netzwerktechnologien gekennzeichnet sein können und auf IP-Ebene durch das Passieren vieler Router *(Hops)*. Dabei wird jedes Paket in jedem Router zwischengespeichert, der Kopf *(Header)* jedes Paketes ausgelesen, das Paket mit dem eventuell abgeänderten Header wieder neu verpackt und entsprechend der aktuellen Routinginformation des Routers weitergeleitet. Auf Netzwerkebene ist dieser Aufwand noch höher: Da hier verschiedene Technologien eingesetzt werden und die Granularität des Netzes höher ist als auf IP-Ebene, gibt es potentiell noch mehr Netzwerkentitäten, auf denen die Pakete zwischengespeichert und vermittelt werden und auch Umsetzungen auf andere Netzwerkprotokolle auftreten können. Beim Pakettransport addieren sich die Verarbeitungszeiten in den Knoten mit den reinen Übertragungszeiten auf den Netzwerkteilstrecken zu einer verbindungsabhängigen Gesamtverzögerungszeit, die auch als *Latenzzeit* bezeichnet wird.

Zum Glück ist man als Architekt einer Middleware von dieser Komplexität insoweit abgeschirmt, als dass man i.d.R. auf Anwendungsschicht arbeitet und somit direkt die Dienste der Transportschicht über Sockets in Anspruch nimmt.

Man braucht also nur die IP-Adresse des Zielsystems und die Portnummer des Prozesses zu kennen, mit dem man eine Verbindung aufbauen will, und schon kann man kommunizieren. Alle Verbindungsdetails erledigen die unteren Schichten in transparenter Weise.

Verbindungen auf Anwendungsebene entlasten den Programmierer zwar von einer Vielzahl komplexer Vorgänge in den Netzwerkschichten, entkoppeln ihn und seine Anwendung aber auch von deren Informationen wie Fehlerraten, Bandbreiten, zugesicherten Eigenschaften und anderen Qualitätsdaten *(Quality of Service - QoS)*, der Art des Interfaces und damit impliziten Annahmen über die Charakteristik der Netzwerkverbindung usw. Messungen von Verbindungsqualitäten auf Anwendungsebene beschränken sich deshalb auf die Ende-zu-Ende-Verbindung und damit auf die Messgrößen Datenübertragungsrate und Latenzzeit auf Transportdienstniveau.

Große Entfernungen machen sich auf Anwendungsebene durch Verzögerungen *(Latenzen)* bemerkbar, welche die synchrone Kommunikation stören oder sogar, unter Beachtung der inhärenten Unzuverlässigkeit von Weitverkehrsnetzen, unmöglich machen. Eine asynchrone Kommunikation kann daher bei geographisch großflächig verteilten Middleware in Bezug auf Leistung und Zuverlässigkeit von Vorteil sein.

3.3.3.3 Administrationsskalierbarkeit

Ein Benutzer gehört mit seinem vernetzen Computer einem administrativen Bereich *(Domäne, domain)* im verteilten System an, der ihm gewisse Rechte und Pflichten zur Nutzung (und eventuell Zahlung) der Ressourcen des Systems gewährt. Voraussetzung dafür ist, dass der Nutzer und der Computer der Domäne bekannt sind, woraus sich auch ein gewisses Vertrauensverhältnis zwischen Administrator (Organisation) und Nutzer ergibt. Dieses Vertrauen besitzt ein Benutzer beim Nutzen von Ressourcen in einer anderen Domäne des verteilten Systems nicht automatisch.

Grundsätzlich besteht ein Sicherheitsbedürfnis in zwei Richtungen: Eine neue Domäne/ein neuer Nutzer im verteilten System wird sich gegen feindliche Angriffe aus dem bestehenden System schützen wollen, und das verteilte System bzw. die darin vorhandenen Domänen werden sich gegen Angriffe aus der neuen Domäne schützen. Erreichen kann man dies über Sicherheitsrichtlinien *(policies)* in jeder Domäne, die domänenfremden Benutzern weniger Rechte und Ressourcen zuteilen als bekannten, vertrauenswürdigen Benutzern. Besonders schwierig ist dieses Thema im Bereich des mobilen Codes (z.B. mobile Agenten), da hier die Ausführung von fremdem Programmcode noch schneller und stärker zu Schäden am Gastsystem führen könnte. Fremder mobiler Code jeglicher Art sollte deshalb immer in einer Art Sandbox ausgeführt werden, um Übergriffe auf das Gastsystem zu verhindern. Als Sandbox bezeichnet man ei-

ne abgeschlossene Ausführungsumgebung für nicht vertrauenwürdigen fremden Programmcode, welche nur einen durch eine Sicherheitsrichtlinie beschriebenen, beschränkten Zugriff auf Ressourcen des Computersystems gewährt und damit nichtgewünschte oder schädliche Zugriffe auf Ressourcen verhindert.

Feingranulare Sicherheitsrichtlinien sind für ein großes verteiltes System sicherlich unerlässlich, können aber auch eine konsistente Nutzersicht sowie die Transparenz beim Ressourcenzugriff behindern.

3.3.3.4 Klassische Skalierungstechniken

Skalierungsprobleme in verteilten Systemen treten in erster Linie als Leistungsprobleme, hervorgerufen durch begrenzte Kapazitäten von Kommunikationsnetzwerken und Ressourcen, auf. Deshalb gibt es nach Neuman [Neu94] nur drei grundsätzliche Techniken zur Skalierung:

- Replikation,
- Caching und
- Verteilung.

Durch Replikation, also das Anlegen von Kopien von Ressourcen, erreicht man eine Lastverteilung, da nun mehrere Dienstzugangspunkte vorhanden sind. Bei guter geographischer Verteilung der replizierten Ressourcen erreicht man sogar eine Verringerung der Latenzzeiten, da die Entfernung zwischen Ressource und Nutzer verringert wird. Als spezielle Ausprägung der Replikation kann das Caching angesehen werden, da hierbei die Ressource nicht vom Dienstanbieter, sondern vom Nutzer repliziert wird. Da nach der Replikation jede Kopie der Ressource ein Eigenleben führt und sich gegebenenfalls verändern kann, können Konsistenzprobleme entstehen, deren Behebungsaufwand die Vorteile wieder aufheben können. Besonders bei einer hochdynamischen Umgebung machen diese Techniken wenig Sinn.

Die Verteilung einer Ressource oder eines Dienstes über das gesamte verteilte System hilft ebenso, Single-Point-of-Failure Konstellationen und punktuelle Überlasten zu vermeiden. Dazu muss eine Ressource oder ein Dienst aber auch in möglichst disjunkte Teilmengen zerlegbar sein. Die Gesamtleistung eines verteilten Dienstes wird dann über mehrere verteilte Teile des Dienstes schrittweise erbracht. Ein gutes Beispiel für einen verteilten Dienst ist das *Domain Name System (DNS)* im Internet, welches logische Rechnernamen in ihre IP-Adressen auflöst.

3.4 Zusammenfassung

Von den bekannten Organisationsstrukturen zur Unterstützung verteilter Anwendungen ist das Middleware-Konzept vielversprechend. Moderne Middleware stellen heute eine große Menge von speziellen Diensten mit Eigenschaften zur Verfügung, die über die Angebote der Dienste von Netzwerkbetriebssystemen weit hinausgehen. Durch diese komplexen Dienste und Kommunikationsprotokolle werden neue Funktionalitäten anwendungsunabhängig angeboten und ermöglichen verteilten Anwendungen eine Kommunikation auf höherer logischer Ebene, wodurch deren Entwickler von der Komplexität der Netzwerkkommunikation entlastet werden.

Die zentralen Konstruktionsziele von Middleware wie Transparenz, Offenheit und Skalierung sind in jedem Falle beim Bau einer Middleware einzuhalten. Es sind aber wegen der Abhängigkeiten dieser Parameter voneinander nicht alle gleichermaßen gut erfüllbar. Je nach Anforderungen der potentiellen verteilten Anwendungen, die die Middleware nutzen sollen, muss auf bestimmte Kerneigenschaften fokussiert werden. Manche Teilaspekte können beim Entwurf einer Middleware sogar im Widerspruch zu den gewünschten Eigenschaften stehen. Für das in der Einleitung, Abschnitt 1.3, andiskutierte Szenario des mobilen Agenten, der zur Lösung seiner Aufgabe Dienste im Netzwerk sucht und die Route zum Ablaufen der Dienste optimieren will, könnte eine Verteilungs- und Ortstransparenz abträglich sein.

Von den Skalierungseigenschaften haben vor allem die Mengen- und die geographische Skalierbarkeit Auswirkungen auf die Leistungsfähigkeit einer zu entwerfenden Middleware. Sie bedürfen daher einer besonderen Beachtung. Zur Lösung eventuell auftretender Skalierungsprobleme besteht die Möglichkeit, mittels Replikation, Caching und Verteilung darauf Einfluss zu nehmen. Mit ausschließlich zentralen oder ausschließlich dezentralen Ansätzen kann man Probleme der Mengenskalierbarkeit unter Internetbedingungen nur sehr schwer lösen.

4 Architekturmodelle verteilter Systeme

Durch das Architekturmodell eines verteilten Systems wird die Verteilung seiner Komponenten und die Beziehungen zwischen ihnen festgelegt. Dabei können die Komponenten verschiedene Aufgaben fest an sich binden, dynamisch annehmen oder mehrere Aufgaben simultan übernehmen. Daraus können sich wiederum wechselnde Beziehungen zwischen den Komponenten ergeben. Die beiden wichtigsten Architekturen, Client-Server-Modell und Peer-to-Peer-Modell, wurden in den bisherigen Kapiteln schon angesprochen. Neu ist in diesem Kapitel das Konzept der mobilen Agentensysteme.

Abschnitt 4.1 stellt das Client-Server-Modell detailliert vor. Im Abschnitt 4.2 wird das Peer-to-Peer-Modell vorgestellt, charakterisiert und der State-of-the-Art konkreter Systeme diskutiert. Nach einem ähnlichen Ablauf werden in Abschnitt 4.3 mobile Agentensysteme behandelt. Im Unterabschnitt 4.3.3 wird eine Einteilung mobiler Agentensysteme nach verschiedenen Entwicklungsstufen vorgenommen, wovon die Stufe 2 für diese Dissertation besonders wichtig ist. Im nachfolgenden Unterabschnitt 4.3.4 werden Anforderungen an die Infrastruktur für ein mobiles Agentensystem der Stufe 2 abgeleitet, anhand derer die aktuellen Standards im Bereich Agenten im Unterabschnitt 4.3.5 und konkrete mobile Agentensysteme im Unterabschnitt 4.3.6 untersucht werden. Die erarbeiteten Ergebnisse dienen im nachfolgenden Kapitel 5 zur Vorstellung des Lösungsvorschlages für eine Infrastruktur für ein mobiles Agentensystem der Stufe 2 und für die Aufstellung der Thesen dieser Dissertation.

4.1 Client-Server-Modell

Das heute vorherrschende Architekturmodell im Internet ist das Client-Server-Modell. Es ist durch eine feste Verteilung der Ressourcen gekennzeichnet, aus der sich eine feste Rollenverteilung und ein festes Interaktionsmuster (Abbildung 4.1) zwischen den an der Kommunikation teilnehmenden Computern ableitet. Die im Netzwerk verteilten Ressourcen werden ausschließlich auf den *Servern* als Dienste angeboten, welche die Rolle der Dienstanbieter einnehmen. Server sind meist auch hardwareseitig besonders ressourcenstarke, spezialisierte Computer, welche auf massenhafte Dienstanfragen und deren Verarbeitung gut reagieren können. Die *Clients* nehmen die feste Rolle der Dienstnutzer ein, die selbst keine Dienste anbieten. Es wird davon ausgegangen, dass ein Client-Computer im

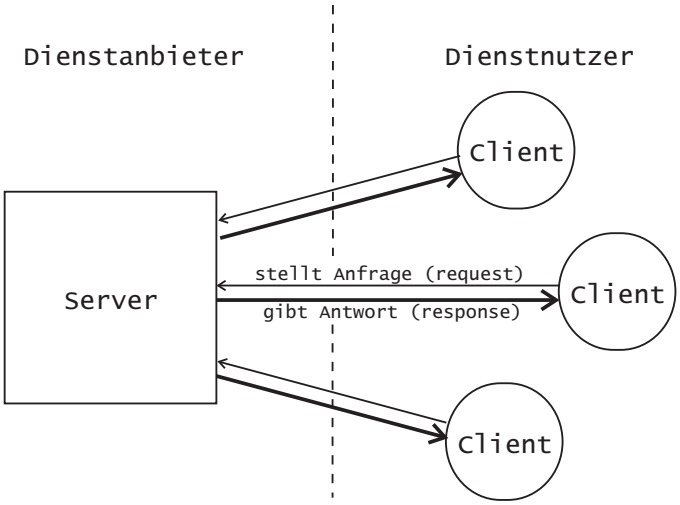

Abbildung 4.1: Interaktionsmuster und Rollenverteilung im Client-Server-Modell

Vergleich zum Server erheblich weniger Ressourcen hat und deshalb auch nur eine begrenzte Verarbeitungskapazität besitzt. Die Annahme einer solchen Ressourcenverteilung impliziert auch eine asynchrone Lastverteilung einer virtuellen Kommunikationsverbindung im Internet. Es wird davon ausgegangen, dass ein Client nur wenig Daten (z.B. Anfragen) an einen Server sendet, dagegen aber sehr viele Daten von Servern herunterlädt. Aus diesem Grund sind z.B. fast alle in Deutschland für Privatkunden erhältlichen DSL-Anschlüsse der Telekommunikationsfirmen asymmetrisch ausgelegt, d.h. die Downloadrate verhält sich zur Uploadrate etwa im Verhältnis 1:10.

Das Client-Server-Modell lässt sich relativ einfach administrieren, da Dienste zentral vom Server angeboten werden. Der Server hat eine bestimmte Netzwerkadresse und befindet sich in einem Netzwerk, welches zur Administrationsdomäne des Eigentümers/Betreibers gehört. Sollen Diensteigenschaften des zentralen Servers geändert werden, wirkt sich dies direkt auf alle Clients, die den Dienst nutzen, aus. Das Dienstangebot ist daher für alle Clients immer konsistent.

Natürlich bringt diese zentralistische Lösung auch die typischen Probleme zentraler Ansätze mit sich. Es können Engpässe bezüglich der Verarbeitungskapazität des Servers auftreten, wenn alle Anfragen an eine einzige Maschine gestellt werden. Auch das umgebende Netzwerk, das die Anfragelast an den Server weiterleiten muss, kann zum Flaschenhals werden. Die Ausfallsicherheit des Dienstes ist, wenn er auf einem einzigen Rechner implementiert ist, nicht gegeben *(single-point-of-failure)*. Skalierungstechniken wie Caching und Replikation mildern zwar diese Probleme, können sie aber nicht vollständig verhindern und bringen neue Probleme, wie Konsistenz, mit sich.

Trotz dieser konzeptbedingten Nachteile überwiegt der Client-Server-Ansatz

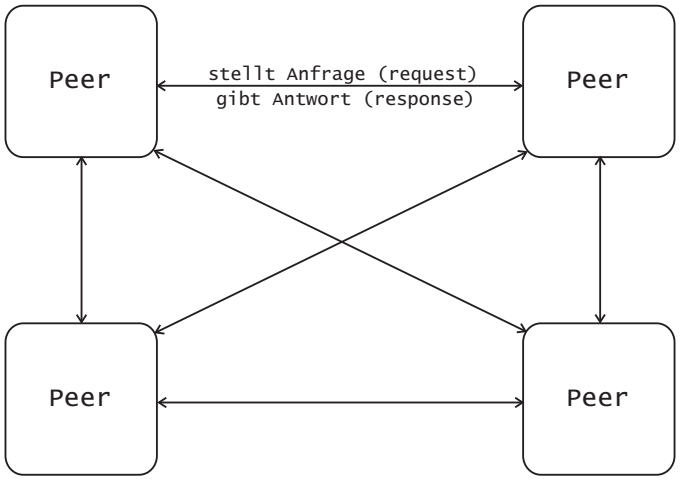

Abbildung 4.2: Beispiel eines Interaktionsmusters und der Rollenverteilung im Peer-to-Peer Modell

in Anwendungsbereichen wie z.B. Online-Banking, Webseiten und E-Commerce, wo Sicherheit, Administrierbarkeit und Konsistenz die wichtigsten Eigenschaften des verteilten Systems darstellen. Das liegt auch daran, dass diese Anwendungen i.d.R statisch sind und die konzeptuellen Probleme in der Praxis durch entsprechend robuste und leistungsstarke Auslegung der Rechnerumgebung in Verbindung mit Kapazitätsplanungen und Lastverteilungsmechanismen recht gut gelöst werden können. Für eine hochdynamische Umgebung mit ständig wechselnden, nicht vorhersagbaren Anforderungen ist dieser Ansatz aber wenig geeignet.

4.2 Peer-to-Peer-Modell

4.2.1 Aufbau und Eigenschaften

Das Peer-to-Peer-Modell ist das ursprüngliche Architekturmodell des Internets. Am Anfang bestand das Internet aus gleichwertigen vernetzten Rechnern mit nicht sonderlich speziellen Aufgaben. Jeder Rechner bot gewisse Dienste an und konnte Dienste auf anderen Rechnern nutzen. Er war bezüglich Ressourcen- und Rollenverteilung Gleicher unter Gleichen, also ein *Peer*[1]. Bei der Interaktion zwischen Peers kann die Rolle als Dienstanbieter und Dienstnutzer ständig wechseln oder auch gleichzeitig ausgeübt werden (vgl. Abbildung 4.2). Peers vereinen also die Funktionalität von Servern und Clients miteinander und werden daher auch als *Servents* [Ste04] bezeichnet.

[1] Der Begriff kommt aus dem Englischen und bedeutet u.a. Gleichgesteller oder Gleichrangiger.

Ein bekanntes, klassisches Beispiel für ein globales Peer-to-Peer-System ist das *Domain Name Service System (DNS)* [Moc87a, Moc87b] des Internets. Zu Beginn des Internets vollzog noch jeder Rechner seine eigene Namensauflösung mit Hilfe einer lokalen Datei, in der die Namen der bekannten Rechner und ihre zugehörigen IP-Adressen untergebracht waren. Die Internetrechner tauschten diese Dateien regelmäßig untereinander aus und aktualisierten ihre eigene Datei um neue oder geänderte Einträge, um eine global konsistente Sicht sicherzustellen. Mit dem zunehmenden Wachstum des Internets war es nicht mehr möglich, alle globalen Einträge in einer einzigen Datei auf jedem Rechner zu verwalten, so dass dafür ein eigener Dienst, der DNS, ins Leben gerufen wurde. Die Aufgabe des DNS-Systems ist die Abbildung logischer Rechnernamen auf deren Internetadressen *(IP)*. Es unterteilt dazu den globalen Namensraum der logischen Rechnernamen in hierarchischer Weise, verteilt die Teile auf verschiedene DNS-Server und bildet daraus eine globale verteilte Datenbank. Mit der hierarchischen Overlaystruktur des DNS-Systems war allerdings auch die Abkehr vom klassischen Peer-to-Peer-Modell besiegelt. Das DNS-System skaliert im globalen Kontext durch seine hierarchische Struktur sehr gut mit der Anzahl der verwalteten Rechnernamen, jedoch nur in einer relativ statischen Umgebung mit immobilen Rechnern.

Allgemein kann an dieser Stelle festgehalten werden, dass das klassische Peer-to-Peer Modell fast entgegengesetzte Eigenschaften im Vergleich zum Client-Server-Modell hat. Durch die Verteilung von Diensten kann ein Flaschenhalseffekt beim Zugriff auf die Dienste über das Netzwerk fast ausgeschlossen werden, ebenso die Überlastung des diensterbringenden Rechners und die Single-Point-of-Failure Problematik. Die Nachteile liegen in der naturgemäß unstrukturierten Vernetzung der Peers, welche auf Anwendungsebene entsprechend der Ende-zu-Ende-Sicht erfolgt und die Ressourcensuche schwierig macht. Jeder Rechner muss mit jedem anderen kommunizieren, um einen Überblick über alle angebotenen Dienste und Ressourcen im Netzwerk zu bekommen. Damit werden bei n Rechnern potentiell $n * (n - 1)$ Verbindungen auf Kommunikationsdienstebene geschaffen, welche nicht mit der potentiell möglichen Anzahl von Rechnern in einem globalen Umfeld skaliert. Auch Sicherheitsaspekte wie Authentifizierung und Autorisierung sind ohne gewisse zentrale Strukturen schwer zu implementieren.

4.2.2 Anwendungsebenen

Das Peer-to-Peer-Modell wurde im Internet zuerst beim DNS System und bei der Routerkommunikation eingesetzt. Nachdem das Peer-to-Peer-Modell in den späten achtziger und den frühen neunziger Jahren des 20. Jahrhunderts ein Nischendasein fristete, brachten es Anwendungen wie Instant-Messaging-Systeme, File-Sharing-Systeme und Grid-Computing Ende der Neunziger Jahre wieder ins

Rampenlicht der Internetwelt. Neu war hierbei, dass das Peer-to-Peer-Modell nun auch auf andere Abstraktionsebenen als auf die grundlegende Internetinfrastruktur angewendet wurde. Die Abstraktionsebenen, auf denen das Peer-to-Peer-Modell eingesetzt werden kann, unterscheidet man auch in [AH02] (beginnend mit der höchsten Abstraktionsebene):

Benutzerebene: Die Benutzer eines Systems verhalten sich wie Peers, nehmen also beide Rollen als Anbieter und als Nachfrager ein. Ein Beispiel ist hier die Onlineverkaufsplattform Ebay. Obwohl das System selbst kein Peer-to-Peer-System ist, interagieren die Benutzer wie Peers.

Dienstebene/Anwendung: Hierunter fallen u.a. die File-Sharing-Systeme, bei denen Daten von einem Peer direkt zum anderen ausgetauscht werden. In der Praxis kann dem Benutzer hierbei, je nach Auslegung des Systems, die Verteilung der Daten sichtbar gemacht werden (z.B. die Information, von welchem anderen Peer die Daten heruntergeladen werden).

Datenzugriffsebene: Hierunter fällt die Suche nach Ressourcen, z.B. Diensten oder Daten. Auf dieser Ebene werden in den meisten File-Sharing Systemen spezifische Strukturen zur Verwaltung von Informationen eingesetzt, um Ressourcen schneller, vollständiger oder mit wenig Aufwand (Netzwerklast) ausfindig zu machen.

Netzwerkebene: Auf dieser Ebene wird das Peer-to-Peer-Modell eingesetzt, um grundlegende, anwendungsunabhängige Netzwerkdienste wie Paket-Routing oder Adressenauflösung auf Netzwerkebene zu realisieren.

Der Einsatz des Peer-to-Peer-Modells kann interessanterweise auf jeder dieser Ebenen unabhängig von den anderen Ebenen erfolgen und daher auch kombiniert werden. Am bekanntesten sind sicherlich die File-Sharing Systeme, die das Peer-to-Peer-Modell auf Dienstebene verwenden und heute oft synonym für den Begriff Peer-to-Peer verwendet werden. Dabei versuchen die meisten modernen File-Sharing-Systeme, die Nachteile der klassischen Peer-to-Peer-Systeme zu überwinden, indem sie durch Overlaystrukturen auf der Datenzugriffsebene die Peer-to-Peer-Netzwerke logisch strukturieren. Über die Overlaystrukturen werden Informationen bezüglich angebotener Ressourcen verwaltet und der Suchaufwand gegenüber dem unstrukturierten Peer-to-Peer-Netzwerk stark verringert. Nach erfolgreicher Suche wird die Ressource dann direkt nach dem Peer-to-Peer-Muster angesprochen.

Die verwendeten Prinzipien und Methoden zur Overlaystrukturierung und zur Verwaltung und Suche von Ressourceninformationen sind vielfältig und auf den jeweiligen Anwendungsfall spezialisiert. Auf einige interessante Beispiele und ihre Eigenschaften wird in Abschnitt 4.2.4 tiefer eingegangen. Im nächsten Abschnitt

wird versucht, eine Klassifizierung von Peer-to-Peer-Systemen entsprechend der Eigenschaften ihrer Overlaystrukturen vorzunehmen.

4.2.3 Klassifikation und architekturelle Parameter

Die Eigenschaften von Peer-to-Peer-Systemen werden ganz wesentlich von der Architektur ihrer Overlaystruktur beeinflusst. Overlaystrukturen von Peer-to-Peer-Systemen lassen sich laut [HD05] nach drei Konstruktionsparametern klassifizieren:

Strukturierungsgrad: Peer-to-Peer-Systeme können entweder unstrukturiert oder strukturiert sein. Bei strukturierten Systemen speichern Peers lokal Ressourceninformationen von anderen Peers und ermöglichen so eine zielorientiertere Suche, allerdings entsteht auch ein zusätzlicher Aufwand zur Verwaltung dieser Informationen. In unstrukturierten Systemen speichern die Peers nur ihre eigenen Ressourceninformationen, so dass eine Suche sehr aufwändig werden kann. Dafür gibt es keinen Zusatzaufwand zum Verwalten des Systems.

Hierarchiegrad: Peer-to-Peer-Systeme können entsprechend der Rollenverteilung ihrer Peers im Overlay eingeteilt werden. Man kann in flache Peer-to-Peer-Systeme, in denen allen Peers Verwaltungsaufgaben zukommen, und in hierarchische Systeme unterscheiden, bei denen nur manche Peers (sogenannte Super-Peers) Verwaltungsaufgaben wahrnehmen. Der Extremfall ist dabei sicherlich ein Peer-to-Peer-System mit einem einzigen zentralen Informationsserver, auf dem alle Ressourceninformationen gehalten werden.

Kopplungsgrad: Der Zusammenschluss von Peers in einem Overlaynetzwerk lässt sich auch anhand der Stärke der Einbindung in die Verwaltungsstruktur einteilen. In eng gekoppelten Systemen hat jeder Peer eine bestimmte feste Rolle in der Struktur, so dass ein Ein- und Austreten entweder nur langsam oder nur von wenigen Peers gleichzeitig vonstatten gehen darf. Diese Eigenschaft beschränkt die Reaktionsfähigkeit auf dynamische Prozesse und die Skalierbarkeit von Peers. In lose gekoppelten Systemen hingegen gibt es diese Beschränkungen weniger stark oder nicht.

Die konkreten Ausprägungen dieser Konstruktionsparameter bestimmen eine weitere Eigenschaft von Peer-to-Peer-Systemen:

Skalierbarkeit: Diese Kerneigenschaft von Peer-to-Peer-Systemen wird im Zusammenspiel mit den zuvor genannten Parametern erbracht. Ein niedriger Kommunikationsaufwand, eine hohe Mengenskalierbarkeit von Peers in

Verbindung mit einer hohen zeitlichen Skalierbarkeit bezüglich der verwalteten Informationen führt zu einer hohen Robustheit und Fehlertoleranz eines Peer-to-Peer-Systems. Diese Eigenschaften sind für die Leistungsfähigkeit und Anwendbarkeit einer Peer-to-Peer-Architektur in der Praxis entscheidend.

Im nächsten Abschnitt werden konkrete Beispiele für Peer-to-Peer-Architekturen vorgestellt und bezüglich der hier aufgestellten Klassifikation diskutiert.

4.2.4 Konkrete Peer-to-Peer-Architekturen

4.2.4.1 Napster

In den späten 1990er Jahren belebten die File-Sharing Systeme die Verwendung des Peer-to-Peer-Paradigmas auf Anwendungsebene neu. Das erste bekanntgewordene System war Napster [Shi01], welches als reine Musiktauschbörse konzipiert war und jetzt in seiner ursprünglichen Form wegen Copyright-Verstößen nicht mehr existiert. Jeder Teilnehmer am Napster-Netzwerk konnte Musiktitel in Form von MP3- oder WMA-Dateien auf seinem Computer zur Verfügung stellen und gleichzeitig von ihm gewünschte Titel suchen und downloaden.

Die Overlaystruktur von Napster bestand aus einer zentralen Datenbank, in welcher für jede Datei (Musiktitel) ein Eintrag mit IP-Adressen (Indexierung) von anbietenden Peers gehalten wurde. Die Nutzer mussten eine bestimmte Software auf ihren Computer laden und ausführen, um sich am Napster-System anmelden zu können. Diese Software wurde bei Napster als Client bezeichnet, während das zentrale Verzeichnis als Server bezeichnet wurde. Ein Nutzer konnte nun im zentralen Verzeichnis nach den gewünschten Musiktiteln und ihrem Ort (IP-Adresse) suchen und sie sich dann direkt vom betreffenden Computer herunterladen (siehe Abbildung 4.3). Napsters Overlaystruktur war keine Peer-to-Peer-Architektur, sondern folgte auf Datenzugriffsebene dem klassischen Client-Server-Ansatz. Die Propagation der Musikdaten erfolgte an einen zentralen Server, auf dem dann ein zentraler Suchservice realisiert wurde. Der Peer-to-Peer-Ansatz kam dann nur beim reinen Datenaustausch zwischen den Peers zum Einsatz.

Der zentrale Suchdienst führt zum bekannten Single-Point-of-Failure Problem und zu einem Flaschenhals bei hoher Abfragelast, der sich in erheblichen Leistungsproblemen, die bis zum Ausfall des Dienstes führen können, äußert. Dynamische Veränderungen im Netzwerk, z.B. hervorgerufen durch mobile Endgeräte, führen zu einer hohen Rate an An- und Abmeldevorgängen, die vom zentralen Napster-Server nicht bewältigt werden könnten. Der Nachrichtenaufwand zur Verwaltung der Ressourceninformationen ist dagegen sehr gering, da jeder Peer direkt mit dem Server kommuniziert.

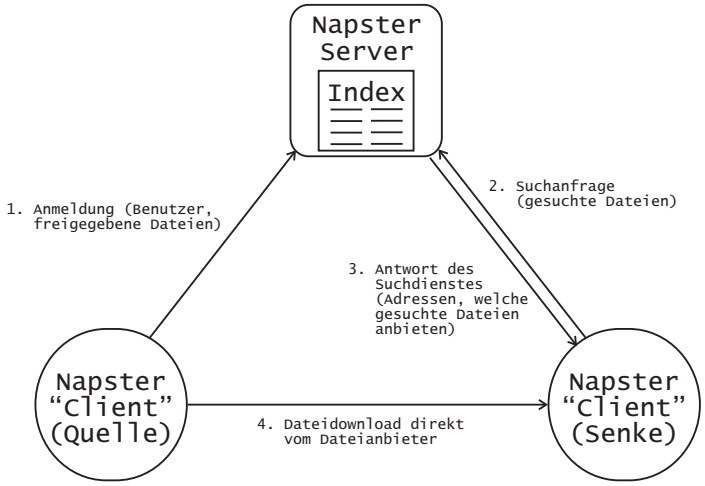

Abbildung 4.3: Architektur von und Kommunikation bei Napster

Napster stellt mit seiner extrem vom Peer-to-Peer-Konzept abweichenden Overlaystruktur einen Sonderfall dar. Möchte man es trotzdem nach den im Abschnitt 4.2.3 aufgestellten Parametern klassifizieren und betrachtet man deshalb den zentralen Server als hochspezialisierten Peer, ist Napster als stark strukturiert, streng hierarchisch, eng gekoppelt und schlecht mengenskalierend einzustufen.

4.2.4.2 Gnutella

Gnutella [Kan01] geht einen anderen Weg als Napster, indem es vollständig dezentral arbeitet. Alle Gnutella Peers sind an der Informationsverwaltung und Dateisuche gleichermaßen beteiligt. Gnutella Peers von vor 2001 [Lim01] nehmen den initialen Kontakt zum Netzwerk über eine Liste von vorher bekannten Peers auf, von denen mindestens einer erreichbar sein muss. Falls dies nicht zutrifft, ist kein Kontakt zum Gnutella-Netzwerk möglich. Gnutella Peers in der Version von 2003 [Lio04] besitzen eine vorkonfigurierte Liste mit hochverfügbaren Peers, welche zur Kontaktaufnahme mit dem Gnutella-Netzwerk dient. Beim Start des Peers versucht dieser eine Anfrage an einen dieser hochverfügbaren Peers, welcher dann eine Liste mit kontaktierbaren Peer zurückschickt. Diese Listen werden durch Mithören des Gnutella-Netzwerkverkehrs gefüllt, denn alle Peers vermitteln Suchanfragen weiter. Somit ist immer eine Liste mit aktuell laufenden Peers garantiert.

Suchanfragen werden im Gnutella-Netzwerk durch Broadcast-*Ping*-Nachrichten versendet. In Gnutella wird dafür, wie für alle anderen Verbindungen auch, eine TCP-Verbindung verwendet. Der Broadcast ist hier logisch gemeint und bezieht sich auf ein Weiterleiten der Ping-Nachricht an andere Peers, also ei-

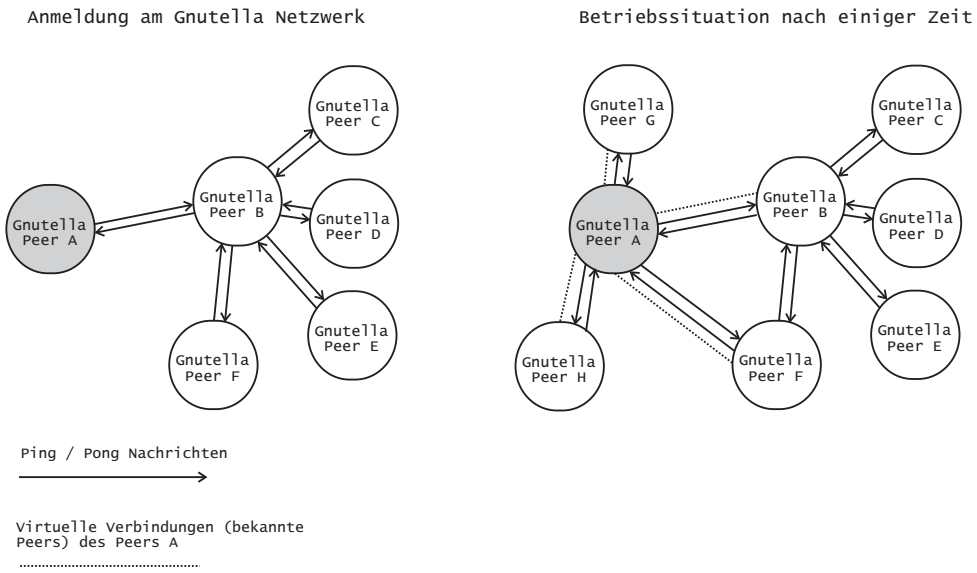

Abbildung 4.4: Links: Anmeldevorgang des Gnutella-Peers A, rechts: Peer A hält nach einiger Zeit vier direkte Verbindungen zu anderen Peers

gentlich mehrere TCP-Verbindungen zu mehreren Peers gleichzeitig. Angepingte Peers senden an den Absender eine *Pong*-Nachricht zurück und "broadcasten" dann an alle ihrerseits bekannten Peers die Nachricht des suchenden Peers weiter. Jeder über diesen Mechanismus erreichte Peer "pongt" nun dem suchenden Peer zurück, so dass dieser immer neue Peers kennenlernt. Jeder Gnutella-Peer hält typischerweise zu bis zu vier anderen Peers Verbindungen aufrecht, wobei diese Peers ausgewechselt werden können, wenn neue Pong-Informationen eintreffen, siehe Abbildung 4.4. Das Gnutella-Netzwerk bildet dadurch einen zufälligen Graphen, in welchem Schleifen nicht ausgeschlossen werden können. Daher überprüft jeder Peer die Ping-Nachrichten auf Wiederholung und versendet diese dann nicht weiter. Um mit einer einzigen Nachricht nicht das ganze Gnutella-Netzwerk zu fluten, wird die Reichweite der Pakete durch einen TTL-Mechanismus beschränkt. Dafür decrementiert jeder Peer beim Eintreffen einer Nachricht einen in der Nachricht mitübertragenen Zähler, den *Time-to-Live (TTL)*. Erreicht der Zählerstand Null, wird die Nachricht nicht weiter versendet und verworfen. Die im Gnutella-Netzwerk übliche Reichweite liegt bei sieben.

Für Suchanfragen schickt ein Peer eine Anfragenachricht *Query* nach dem beim Ping beschriebenen Broadcast-Mechanismus los. Jeder Peer, der die Query erhält, durchsucht seine lokal freigegebenen Daten auf Übereinstimmung und antwortet bei Erfolg mit einer *Query-Hit* Nachricht. Der suchende Peer versucht dann, die Datei direkt (außerhalb des Gnutella-Netzwerkes) von einem

antwortenden Quellen-Peer per *HTTP-Get* zu laden. Falls der Quellen-Peer hinter einer Firewall sitzt und eine HTTP-Get Anfrage von außen nicht möglich ist, sendet der suchende Peer eine *Push*-Nachricht über das Gnutella-Netzwerk an den Quellen-Peer, der darauf hin eine Verbindung zum suchenden Peer durch die Firewall hindurch öffnet und ihm ein HTTP-Get ermöglicht.

Gnutella ist aus Anwendersicht ein einfaches und effektives Protokoll zum Datenaustausch. Es hat quasi kein Overlaynetzwerk und ist daher unstrukturiert, flach organisiert und besitzt einen geringen Kopplungsgrad. Durch diese Eigenschaften ist es sehr fehlertolerant. Gnutella fängt eine gewisse Netzwerkdynamik (in Bezug auf das Hinzufügen und Verschwinden von Peers) ab und kann seine Topologie den Erfordernissen anpassen. Diese Vorteile werden aber teuer erkauft, denn die Netzwerklast für Suchanfragen und Netzwerkverwaltung ist im Gnutella-Netzwerk immens hoch. Die verwendete Art von logischem Broadcast über TCP ist zwar auch über ein globales Netzwerk zuverlässig, erzeugt aber eine sehr hohe Zahl an Nachrichten. Die Last inklusive Antworten berechnet sich bei einer typischen, durchschnittlichen Anzahl von V Verbindungen pro Peer wie folgt [APHS02]:

$$2 * \sum_{i=0}^{TTL} V * (V-1)^i.$$

Es entstehen also z.B. für eine TTL von sieben und V von vier 26240 Nachrichten im Netzwerk. Dieser Kommunikationsaufwand entsteht bei jedem Eintritt von Peers in und Austritt von Peers aus dem Gnutella-Netzwerk sowie bei Suchanfragen. Daher skaliert Gnutella nicht mit der Menge an entstehenden Nachrichten bei höherer Netzwerkdynamik.

Neuere, leistungsfähigere Gnutella Versionen verwenden u.a. Caching, um diesem Problem zu begegnen und die Netzwerklast bei den Suchanfragen zu reduzieren. Doch diese Versionen kommen auch nicht mehr ohne gewisse spezialisierte Peers aus, welche zusätzliche Verwaltungsfunktionen erfüllen. So gibt es ab der Version 2003 [Lio04] *Ultranodes*, welche als erste Kontaktpunkte für die Anmeldung am Gnutella-Netzwerk dienen und gleichzeitig einen *Backbone* aus leistungsstarken, hochverfügbaren Netzwerkknoten mit stabiler und hoher Netzwerkbandbreite bilden. Der Backbone aus Ultranodes bildet einen festen Infrastrukturteil. Somit entsteht eine hierarchische Infrastruktur mit engerem Kopplungsgrad, die einige Vorteile der ursprünglichen Gnutella-Idee ad absurdum führt. Es existieren mittlerweile viele verschiedene Gnutella-Peer-Implementierungen wie Bearshare [Bea05], LimeWire [Lim05] und Morpheus [Mor05] (ab 2003). Die Overlaystrukturen aller dieser Implementierungen sind als strukturiert, hierarchisch und eng gekoppelt einzustufen.

4.2.4.3 FastTrack

FastTrack wird oft als Peer-to-Peer-Protokoll bezeichnet, stellt aber ein Peer-to-Peer-System zum Datenaustausch dar. Es unterscheidet drei Arten von Netzwerkknoten: *Normale Peers*, *Super-Peers* und *Super-Super-Peers*. Normale Peers bieten nur Daten an und laden diese von anderen Peers, sie haben keine besonderen Aufgaben bei der Netzwerkverwaltung. Super-Peers sind normale Peers, die zusätzlich Indexinformationen verwalten. Der initiale Einstieg von normalen Peers ins FastTrack-Netzwerk erfolgt ähnlich wie bei den moderneren Gnutella Versionen über hochverfügbare Super-Peers, welche im FastTrack-Vokabular als Super-Super-Peers bezeichnet werden. Diese führen die Listen mit aktuellen Super-Peers. Es existieren also Peers mit verschiedenen Rollen, womit FastTrack kein reines Peer-to-Peer-System im ursprünglichen Sinne darstellt. Die hochverfügbaren Super-Super-Peers bilden einen festen Bachbone als zuverlässige Kerninfrastruktur. So gesehen kombiniert FastTrack die Vorteile von Napster und Gnutella, wobei gegenüber Napster von einer vollständig zentralen Indexierung abgegangen wurde und die Indexe auf mehrere Super-Peers verteilt werden.

Super-Peers verwalten die Datenindexe der Peers, welche sich bei ihnen angemeldet haben, halten Verbindungen zu anderen Super-Peers und leiten Suchanfragen im FastTrack-Netzwerk weiter. Normale Peers können mit mehreren Super-Peers verbunden sein, an die sie ihre Suchanfragen stellen. Können die direkt verbunden Super-Peers die Suchanfrage nicht beantworten, leiten sie die Suchanfrage an andere Super-Peers weiter. Dabei wird wie bei Gnutella ein TTL-Mechanismus verwendet. Der Download von gefundenen Dateien erfolgt wie bei Napster direkt zwischen dem suchenden und dem nachfragenden Peer.

Die Super-Peers bilden wie im moderneren Gnutella-Netzwerk eine gewisse zentrale Infrastruktur, die aber, bis auf die Super-Super-Peers, nicht fest ist. Die Super-Peers können entsprechend ihrer Leistungsfähigkeit dynamisch gewählt werden.

Der strukturierte, hierarchische, eng gekoppelte Organisationsansatz von FastTrack kann bei einer ausgewogenen Anzahl von Super-Peers und Peers mengenskalieren, falls die Reorganisation des Netzwerkes schnell genug erfolgt. Da jedoch vom Benutzer die Wahl des eigenen Computers zum Super-Peer explizit unterdrückt werden kann, ist die Skalierbarkeit nicht sicher gewährleistet. Der Nachrichtenaufwand zum Eintreten in und Austreten aus dem FastTrack-Netzwerk und der Suche nach Ressourcen ist geringer als bei Gnutella, jedoch höher als bei Napster. Die Robustheit und Skalierungsfähigkeit liegt ebenfalls zwischen Napster und Gnutella. Die genauen Leistungsausprägungen sind vom aktuellen Konstellationszustand des FastTrack-Netzwerkes abhängig.

Das FastTrack-Netzwerk ist bis heute sehr erfolgreich. Es existieren zahlreiche Peer-Implementierungen, darunter Kazaa [Kaz05], Grokster [Gro05], Mor-

pheus [Mor05] (bis 2002, danach ein Gnutella-Clone) und MLDonkey [Mld05]. Speziell MLDonkey unterstützt dabei mehrere Peer-to-Peer-Netzwerke auf einmal und kann z.B. gleichzeitig ein FastTrack- und ein Gnutella-Peer sein.

4.2.4.4 Freenet

Freenet [CSWH01] stellt einen interessanten Ansatz zur Organisation eines Peer-to-Peer-Systems dar, ist allerdings stark auf die anonyme Publikation und das Suchen von Daten spezialisiert. Es folgt, wie Gnutella, dem reinen Peer-to-Peer-Ansatz, so dass für den Zugang zum Netzwerk schon mindestens ein Freenet-Peer bekannt sein muss. Zur Wahrung der Anonymität werden Datenschlüssel als Suchindex für die gesuchten Daten verwendet und auch die Daten selbst nur verschlüsselt übertragen. Der Suchindex wird aufgeteilt und über alle Peers verteilt.

Freenet verwendet einen adaptiven Routingmechanismus, der Suchanfragen in Richtung des wahrscheinlichsten Speicherortes der gesuchten Ressourceninformation leitet. Dazu führt jeder Peer eine Routingtabelle mit Einträgen, die aus dem Datenschlüssel, der TCP-Verbindung des Peers, wo das Datum gespeichert ist, und eventuell auch den verschlüsselten Daten selbst bestehen. Die Routingtabelle wird mit Hilfe von empfangenen Suchanfragen und Antworten gefüllt, so dass ein Peer nach und nach mehr Wissen über die Netzwerksituation erlangt und die Inhalte der Routingtabelle gleichzeitig an diese adaptiert werden. Eine empfangene Suchanfrage wird zuerst mit den eigenen Schlüsseln verglichen. Kann die Nachfrage nicht befriedigt werden, sendet der Freenet-Peer die Suchanfrage an genau den Peer seiner Routingtabelle weiter, dessen Datenschlüssel den geringsten lexikographischen Abstand zum gesuchten Schlüssel hat und dessen Daten dem gesuchten Datum am ähnlichsten ist. Somit routen Freenet-Peers ihre Suchanfragen wesentlich gezielter und effizienter durch das Peer-to-Peer-Netzwerk, als es bei Gnutella der Fall ist. Zur Laststeuerung kommt ebenfalls ein TTL-Mechanismus zum Einsatz.

Abbildung 4.5 zeigt die Suche in Freenet: Peer A sucht Datei ”target” und schickt die Suchanfrage entsprechend seines Routingtabelleneintrages an Peer B. Dieser findet die Datei in seinen Daten nicht, decrementiert die TTL der Suchanfrage um eins und schickt sie weiter an Peer C, der den Schlüssel mit der geringsten lexikographischen Distanz zur Anfrage besitzt. Auch Peer C findet die Datei nicht und meldet dies an Peer B. Peer B decrementiert die Suchanfrage erneut und schickt sie an Peer D, welcher die Datei lokal gespeichert hat. Peer D meldet den Treffer in Form seiner Adresse über Peer B an Peer A weiter. Auf diesem Rückpfad aktualisieren die Peers B und A ihre Routingtabellen. Peer A hat nun die Adresse der gewünschten Datei ”target” und lädt diese direkt von Peer D.

Die gespeicherten Suchanfragen und Antworten führen zu einer ständigen Ak-

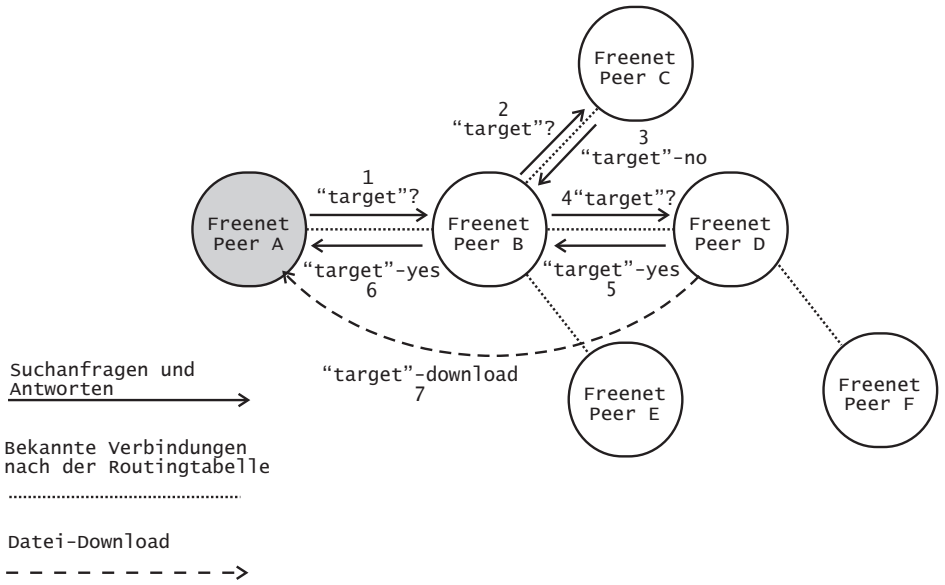

Abbildung 4.5: Routing der Suchanfragen im Freenet-Netzwerk

tualisierung der Routingtabellen, die als Cache dienen. Dabei wachsen diese Tabellen ständig an. Deshalb werden die Größen der Tabellen beschränkt und Einträge nach der *last recently used*-Strategie wieder entfernt. Da Freenet-Peers nicht nur Suchschlüssel, sondern auch verschlüsselte Daten selbst cachen können, werden zuerst die am wenigsten gebrauchten Daten entfernt und ihre Suchschlüssel noch im Cache behalten, bevor auch diese verworfen werden.

Freenet vermeidet die ineffizienten logischen Broadcasts wie in Gnutella und kann trotzdem auf zentrale Knoten verzichten. Die Suchpfadtiefe muss allerdings wesentlich höher sein, da die Suchanfrage immer nur an einen Peer geschickt wird, so dass für die Suchanfragen TTL von bis zu 500 angegeben werden. Da auch Freenet nicht garantiert zyklenfrei arbeitet, wird wie bei Gnutella eine eindeutige ID für jede Suchanfrage vergeben, so dass Schleifen aufgelöst werden können.

Das Cachen von Daten führt nach einiger Betriebszeit zu einer signifikanten Reduktion der Suchpfadlängen (*Small-World Phänomen* [Kle99]), aber auch zu einer Datenspezialisierung der Peers, denn je öfter ein Datum nachgefragt wird, desto häufiger wird seine Suchantwort und/oder das Datum selbst in den Routingtabellen gecacht. Graphen mit Small-World Verhalten sind genau durch solch ein starkes Clustering gekennzeichnet. Die Untersuchungen zu diesem Verhalten stammen ursprünglich aus der Soziologie und bezeichnen Graphen, in denen Knoten, die mit einem bestimmten anderen Knoten verbunden sind, meist auch untereinander verbunden sind, wie bei soziologischen Netzwerken zwischen

Menschen. Auch die Freenet-Peers verhalten sich so nach einiger Zeit, so dass die Suchpfadlänge P bei n Knoten und k Verbindungen (Routingeinträgen) pro Knoten analog wie in den Small-World Untersuchungen [NSW01] auf

$$P = log(n)/log(k)$$

reduziert wird.

Die Anonymität der Benutzer und ihrer veröffentlichten Daten standen beim Entwurf des Freenet-Systems mit an erster Stelle. Die Realisierung der Suchschlüssel über Hashfunktionen stellt diese Anonymität zwar sicher, hat bei der Ähnlichkeitssuche aber starke Nachteile. Dateinamen und -pfade, die sich stark ähneln, aber nicht exakt die gleiche Zeichenfolge einhalten (z.B. teilweise enthaltene Pfadangaben zu Dateien), führen schon zu so unterschiedlichen Hash-Werten, dass die Datei nicht gefunden werden kann. Die textuelle Suche in Gnutella mit Strings und Teilstrings ist für den Anwender wesentlich leichter und durchschaubarer.

Freenet benutzt ein strukturiertes (verteilter Index), flach organisiertes (nur gleichwertige Peers) Overlaynetzwerk. Die Peers sind lose gekoppelt und verfügen über ein Lernverhalten, so dass das Wissen über das Netzwerk mit der Anwesenheit eines Peers im Netzwerk wächst. Freenet verwendet Caching und Replikation zur Verbesserung der Skalierbarkeit. Die Suchanfragen werden immer nur einzeln gestellt, wodurch die Nachrichtenlast gering gehalten wird. Die Mengen- und Kommunikationsskalierbarkeit ist daher hervorragend. Obwohl die Suche sehr zielgerichtet verläuft, kann es gerade bei neu in Freenet eingetretenen Peers vorkommen, dass die Suchanfragen relativ lange unterwegs sind, bevor sie beantwortet werden, da erst nach zunehmender Interaktion mit dem Netzwerk das Small-World Phänomen zu wirken beginnt.

4.2.4.5 P-Grid

P-Grid [Abe01] ist ein weiteres Beispiel für ein völlig dezentral arbeitendes Peer-to-Peer-System, welches aber auf einem virtuellen verteilten binären Suchbaum basiert. Im binären Suchbaum stellt jeder Peer ein Blatt dar und speichert Informationen über einen genau definierten Teilbereich des Baumes. Der Ort des Peers im Baum entspricht dem Binärsting, welcher den Pfad zu ihm beschreibt.

Jeder Peer ist für einen bestimmten Bereich an Informationen zuständig. Die Informationen sind binär verschlüsselt, d.h. es gibt eine Abbildung der Strings der Dateinamen auf binäre Strings. Dazu wurden in P-Grid präfixerhaltende Hash-Funktionen verwendet, so dass gilt [APHS02]:

$$s1\ präfix\ s2 \Longrightarrow Schlüssel\ (s1)\ präfix\ Schlüssel\ (s2).$$

Jeder Peer hält nun einen Informationsspeicher mit Informationen über die Dateien, für die er verantwortlich ist. Dies sind alle Informationen, welche mit

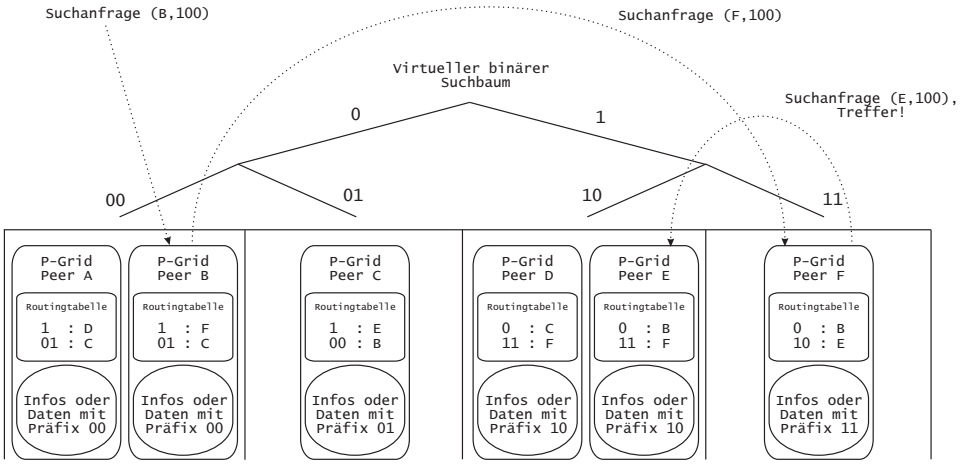

Abbildung 4.6: Suchen im P-Grid

einem Präfix beginnen, der dem Pfad des Peers im Suchbaum entspricht. Ein Peer mit dem Pfad 01001010 ist also für Informationen mit genau diesem Präfix zuständig. Dabei ist es egal, ob der Peer nun die Daten selbst gespeichert hat oder nur die Adressinformationen der Peers, welche die betreffenden Daten speichern, da nur die Adresse des gesuchten Datums für den suchenden Peer wichtig ist, denn die gesuchten Daten werden außerhalb des P-Grid direkt vom Ziel-Peer beschafft. Zusätzlich führt jeder Peer eine Routingtabelle mit Adresseinträgen über andere Peers, welche für Informationen mit anderen Präfixen zuständig sind. Abbildung 4.6 zeigt den Aufbau des und eine Suche im P-Grid. Wird eine Anfrage nach dem Datum 100 an Peer B gestellt, für welches er nicht zuständig ist, sucht er den am weitesten übereinstimmenden Eintrag in seiner Routingtabelle, in diesem Falle der erste Eintrag mit Präfix 1, der auf Peer F zeigt. Dieser ist wiederum für Daten mit Präfix 11 zuständig, weshalb er die Suchanfrage an Peer E, zuständig für den Präfix 10, weiterleitet. Peer E hat nun das passende Datum oder die Adresse des passenden Peers gespeichert. Wie in der Abbildung 4.6 zu sehen ist, speichern mehrere Peers die gleichen Daten und sind unter dem gleichen Suchpfad erreichbar. Durch diese Redundanz wird die Ausfallsicherheit im P-Grid erhöht und sichergestellt, dass immer ein Peer mit den gewünschten Informationen über die gesuchten Daten erreichbar ist.

P-Grid ist ein selbstorganisierendes Peer-to-Peer-Netzwerk. Da keine zentralen Instanzen beteiligt sind, muss der Eintritt ins Netzwerk wie bei den anderen dezentralen Ansätzen durch einen historischen Kontaktpunkt erfolgen, also eine alte gespeicherte Peer-Adresse oder eine fest einprogrammierte Adresse eines zentralen Kontaktpunktes. Beim Kontakt mit dem ersten Peer wird dessen Routingtabelle übernommen oder aber beide beteiligten Peers spezialisieren sich weiter, indem sie den binären Präfix um eine Stelle verlängern und die Daten-

informationen untereinander aufteilen. Diese Form des automatischen Wachsens des P-Grid kann auch durch zufällige Interaktionen zwischen Peers hervorgerufen werden. Für ein ausgewogenes P-Grid muss das Verhältnis von gespeicherten Daten und Suchbaumtiefe in einer gewissen Balance gehalten werden, weshalb eine neue Spezialisierung und damit größere Suchbaumtiefe nur bei einer Überschreitung einer gewissen Datenmenge auf den beteiligten Peers erfolgt. Der Aufwand einer Suche in P-Grid bei n teilnehmenden Peers liegt aufgrund der Binärbaumstruktur bei lediglich $O(log\ n)$ [Abe01]. Obwohl wie in Freenet mit Hash-Werten gesucht wird, kann und muss bei P-Grid auch nach jedem beliebigen Teilstring gesucht werden.

Entsprechend der Klassifikation aus Abschnitt 4.2.3 ist P-Grid ein strukturiertes, flach organisiertes und lose gekoppeltes Netzwerk von Peers. Es besitzt eine vollständig dezentrale Informationsverteilung, braucht keine zentrale Information über die Struktur des Netzwerkes und verwendet nur lokale Interaktionen zwischen in der Overlaystruktur benachbarten Peers. Es unterstützt Informationsupdates und kann mit dynamischen Peer-Adressen umgehen. Die verwalteten Informationen werden zur Verbesserung der Robustheit und Fehlertoleranz zum Teil redundant gehalten. Dynamische Netzwerke werden dadurch sehr gut unterstützt und die Mengenskalierung stellt kein Problem dar. Ebenfalls gering ist der Suchaufwand aufgrund der Binärbaumstruktur.

4.2.4.6 Chord

Chord [DBK+01] ist ein dezentraler Peer-to-Peer-Verzeichnisdienst, der Schlüssel in Form von Hash-Werten verwendet. Dabei werden Daten und IP-Adressen der Peers (IDs) mit der gleichen Hash-Funktion verschlüsselt, so dass sie in den gleichen Wertebereich abbilden. Die Hashwerte der Daten sollen hier als Indexe, die der Peers als IDs bezeichnet werden. Die Peers bilden entsprechend ihrer IDs eine natürliche Reihenfolge im Wertebereich. Die Indexe der Daten werden nun den Peers so zugeordnet, dass jeder Index eines Peers kleiner oder gleich seiner ID ist. Größere Indexe werden dem Peer mit der nächsten ID zugeordnet. Indexe mit einem größeren Wert als der größten ID gehören dann wieder zum Peer mit der kleinsten ID. Durch diese Regelung entsteht ein logischer Ring von IDs und Indexen. In diesem Ring sind alle Indexe vollständig untergebracht.

Für das Auffinden der Daten verwenden die Peers Routingtabellen mit $O(log\ n)$ Einträgen bei n Peers im Ring. Die Einträge enthalten drei Werte:

1. Einen Tabellenindex i mit (Zählnummer),

2. einem Intervall, welches den Bereich zwischen dem nächsten möglichen Knoten $ID + 2^1$ und dem übernächsten Knoten der Tabelle, der sich aus $ID + 2^2$ berechnet, angibt, und

3. dem Nachfolger für das Intervall, die ID des nächsten real existierenden Peers.

Abbildung 4.7 zeigt einen Chord-Ring mit drei Peers und den dazugehörigen Routingtabellen.

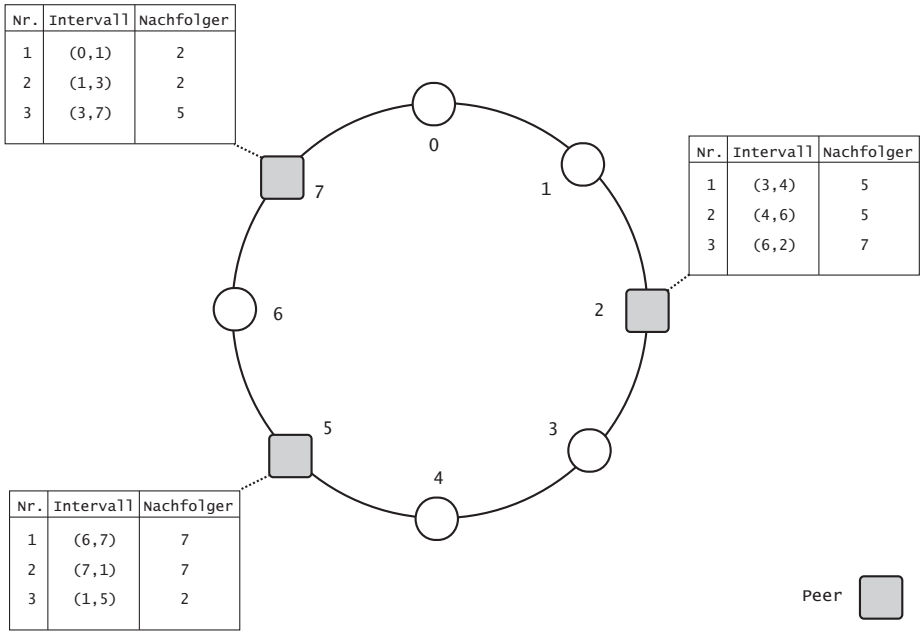

Abbildung 4.7: Aufbau des Chord Rings

Beim Suchen nach einem Index kann ein beliebiger Chord-Peer angesprochen werden. Der Peer vergleicht den gesuchten Index mit seinen Einträgen. Bei negativer Suche wird die Routingtabelle konsultiert. Da die Intervalle der Routingtabelle den gesamten Chord-Ring abbilden, wird einfach das Intervall gewählt, in dem der Index des Datums liegt. Die Suchanfrage wird dann an den zugehörigen Nachfolger (einen realen Peer) weitergeleitet. Der Nachfolger ist definitiv "näher" am gesuchten Index.

Durch die oben beschriebene Auswahl der Intervalle wird garantiert, dass es höchstens $O(log\ n)$ Einträge in einer Tabelle gibt und gleichzeitig maximal $O(log\ n)$ Suchanfragen im Netzwerk versendet werden müssen, um ein Datum zu finden. Die Intervalle vergrößern sich dabei quadratisch, so dass näher an der Peer-ID liegende Indexe schneller gefunden werden als weiter weg liegende. Beim Hinzufügen oder Herausfallen von Peers müssen die Indexe verschoben und Routingtabellen angepasst werden, was mit hoher Wahrscheinlichkeit mit einem Aufwand von $O(log^2\ n)$ möglich ist.

Chord ist wie P-Grid ebenfalls ein strukturiertes und flach organisiertes Peer-to-Peer System. Es ermöglicht eine variable Unterteilung des Schlüsselraumes.

Die Mengenskalierung ist kein Problem. Der Nachrichtenaufwand für die Suche ist gering, für die Aktualisierung etwas höher. Im Gegensatz zu P-Grid verwendet Chord aber zentralisiertes Wissen (jeder Peer besitzt Informationen über den gesamten Chord-Ring), um die Struktur aufzubauen und die Informationen zu verteilen. Dadurch sind die Chord-Peers deutlich enger gekoppelt, da beim Ausfall eines Peers alle anderen ihre Tabelleneinträge aktualisieren müssen. Dynamische Peers werden dadurch weniger gut unterstützt. Bei hohen Aktualisierungsraten kann das Netz zusammenbrechen.

4.2.4.7 JXTA - Eine universelle Peer-to-Peer-Architektur

JXTA wurde ursprünglich von SUN Microsystems entwickelt und ist mittlerweile ein Open-Source Projekt [Jxt01]. Im Gegensatz zu den bisher beschriebenen Peer-to-Peer-Systemen mit proprietären, anwendungsbezogenen Lösungen stellt JXTA eine Architektur dar. Mit JXTA soll eine Peer-to-Peer-Plattform mit universellem Charakter etabliert werden. Zielanforderungen bei der Entwicklung von JXTA waren von Anfang an Plattformunabhängigkeit bezüglich Hardware und Software sowie Interoperabilität [Gon02].

Die dreischichtige JXTA-Architektur besteht aus dem Kern *(JXTA-Core)*, einer Diensteschicht *(JXTA-Services)* und den Anwendungen *(JXTA-Applications)* selbst. Dabei dienen die Kernfunktionalitäten und höheren Dienste der Diensteschicht als Basis und Querschnittsfunktionen für alle Peer-to-Peer-Anwendungen (siehe Abbildung 4.8).

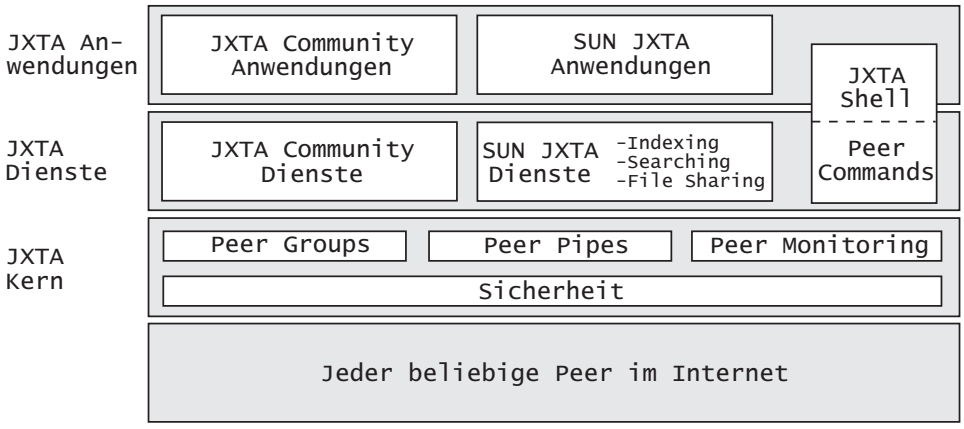

Abbildung 4.8: Die Softwarearchitektur von JXTA, in Anlehnung an [Gon02]

Die JXTA Kerntechnologiekonzepte (JXTA Core) sind der kleinste gemeinsame Nenner in eine JXTA-Umgebung und beschreiben verschiedene Protokolle auf Anwendungsebene, welche u.a. Dienstprimitiven zur Adressierung von Peers und

anderen Ressourcen, zur Peer-Gruppenbildung *(Peer Groups)*, zur Kommunikation *(Peer Pipes)* und zur Peerüberwachung *(Peer Monitoring)* zur Verfügung stellen. Mit Hilfe dieser Protokolle wird das grundlegende Peer-to-Peer-Netzwerk gebildet. Allerdings werden auf dieser Ebene keine neuen Konzepte, sondern vielmehr Technologien verwendet, welche schon aus anderen Peer-to-Peer Systemen und koordinationsbasierten verteilten Systemen bekannt sind, so u.a. logische Peer Gruppen (Gnutella), lokale Broadcasts zur Bekanntmachung (JINI [ASW+99]) und Rendezvous Punkte (Super-Peer-Prinzip, wie z.B. bei FastTrack).

Die JXTA Version 2.0 von 2003 [TAA+03] enthält einige Verbesserungen und Verfeinerungen auf allen Ebenen sowie die Beschreibung einer Referenzimplementierung in J2SE. Es existieren viele Verbesserungen und Anpassungen der JXTA-Plattform, wie etwa *Expeerience* [BCMT03], eine interessante Erweiterung von JXTA auf die Erfordernisse von MANETs.

Interessant ist auch das Leistungsverhalten von JXTA. Messungen der Protokolleffizienz auf verschiedenen Kommunikationsdienstebenen [AHJN04] von JXTA der Version 2.2.1 zeigen gegenüber einer reinen JAVA-Socketverbindung deutliche Leistungseinbußen in Abhängigkeit von der Payloadgröße (der Payload ist die Nutzdatengröße an der Gesamtgröße einer Nachricht). Dies ist nicht verwunderlich, da z.B. auf der Ebene der Peer-Pipes die Mindestgröße eines Paketes schon 877 Bytes beträgt. Trotzdem werden bei der Latenzmessung in einem 100 MBit Fast Ethernet beachtlich gute Werte erreicht. Die Latenz über eine Unicast Pipe beträgt lediglich 1,22 ms, was gegenüber $< 0{,}1$ ms bei einem JAVA-Socket zwar immer noch sehr hoch, aber in Anbetracht der Latenzzeiten in einem globalen Peer-to-Peer-Netzwerk wiederum sehr wenig ist. Die Kommunikationskanäle in JXTA scheinen damit auch für eine verlässliche Leistungsmessung der Verbindungsqualitäten zwischen den Peers geeignet zu sein.

JXTA bietet eine mittlerweile ausgereifte und performante Plattform zur Erstellung eigener Peer-to-Peer-Anwendungen. Auch wenn einige Eigenschaften gegenüber proprietären Peer-to-Peer-Systemen noch zurückliegen, wird es ständig weiterentwickelt und verbessert. Der große Vorteil von JXTA liegt dabei eindeutig in den sauber abgetrennten Funktionsbereichen und der sauberen, gut dokumentierten API.

4.2.4.8 Andere interessante Peer-to-Peer-Systeme

TLS *TLS* [BL04] steht für *Tree-Based DHT Lookup Service* und verfolgt ebenso wie P-Grid und Chord einen binärbaumbasierten Ansatz zur Overlaystrukturierung von Peer-to-Peer-Systemen. Im Gegensatz zu den zuvor genannten Ansätzen liegt der zeitliche Aktualisierungsaufwand zum Einfügen und Heraustreten eines Peers aus dem System bei $O(log\ n)$, wenn n die Anzahl der Peers im Systems ist. TLS teilt den Binärbaum in einen Wald von Bäumen, indem es

Konten einer Binärbaumebene in der Nähe der Wurzel miteinander verbindet. Diese Knoten haben dann Super-Peer Eigenschaften, da sie eine für das Funktionieren des Gesamtsystems hervorgehobene Rolle übernehmen. Operationen, wie das Hinzufügen, Entfernen und Suchen von Knoten, können daher eventuell nur in einem Teilbaum stattfinden und nicht die ganze Struktur beeinflussen. Um die hervorgehobene Rolle der Super-Peers abzusichern, wird in TLS die Verweilzeit im TLS-System betrachtet. Peers mit einer langen "Lebenszeit" können ihre Position im Binärbaum verändern und nach oben klettern. Ab einer gewissen "Lebenszeit" werden sie als für die Rolle eines Super-Peers geeignet betrachtet und können dessen Funktionalität übernehmen. Damit kann sich das TLS-Overlaynetzwerk an sich ändernde Umweltbedingungen adaptieren.

Die Implementierung von TLS wird seit dem Jahr 2004 in Aussicht gestellt, ist bisher aber noch nicht realisiert worden. Das Projekt scheint, obwohl der Ansatz sehr vielversprechend erscheint, nicht mehr weitergeführt zu werden.

INGA *INGA* [Lö05] steht für *Interest based Node Grouping* und strukturiert das Peer-to-Peer-Netzwerk mit Hilfe von Overlays, die auf semantischen Metainformationen über die gesuchten Informationen beruhen. Dadurch wird eine effiziente Suche möglich, da die Menge der relevanten Knoten auf Basis eines themengebundenen Overlays eingeschränkt wird. Dazu trägt auch das schon erwähnte *Small World Phänomen* [Kle99] bei. INGA unterstellt eine systemweit gültige Ontologie, um die Suchinformationen semantisch zu strukturieren. Die Autonomie der Peers wird erreicht, indem einzig lokale Profile verwendet werden. INGA verbindet strukturierte Ansätze beim Overlay mit den unstrukturierten Ansätzen bei der Informationsverwaltung. Der große Unterschied zu anderen Peer-to-Peer-Systemen liegt bei INGA in der Verwendung von semantischen Suchinformationen, mit Hilfe derer die Begriffswelt des Benutzers bis auf die Datenzugriffebene des Overlaynetzwerkes erweitert wird.

4.2.5 Zusammenfassung Peer-to-Peer-Architekturen

Das Peer-to-Peer-Modell kann auf verschiedenen Abstraktionsebenen eingesetzt werden. Die wichtigste ist sicherlich die Ebene der Anwendungsdienste, auf der das Peer-to-Peer-Modell zu einer guten Lastverteilung und Mengenskalierbarkeit führt. Jedoch ist das Auffinden von Ressourcen in einem an sich strukturlosen Peer-to-Peer-Netzwerk ein heikles Problem. In der Praxis werden daher alle Peer-to-Peer-Systeme durch Overlaystrukturen unterstützt, welche als Infrastruktur für das Auffinden verteilter Ressourcen dienen.

Es existieren die unterschiedlichsten Ansätze und Strategien, um Overlaynetzwerke zu verwalten und Ressourcen darin zu lokalisieren. Diese verschiedenen Peer-to-Peer-Architekturen lassen sich mit Hilfe der Parameter Strukturierungs-

grad, Hierarchiegrad, Kopplungsgrad und Skalierung charakterisieren. Die Overlaystrukturen global verteilter, mengenskalierender Peer-to-Peer-Architekturen, die eine gewisse Netzwerkdynamik beherrschen können, sollten folgende Eigenschaften besitzen:

1. Eine flache Hierarchie mit gleichwertigen Peers und möglichst keine hervorgehobenen Netzwerkteilnehmer, deren Ausfall große Auswirkungen auf das Gesamtsystem haben würde.

2. Einen geringen Kopplungsgrad, damit Peers weitgehend autonom agieren können, ohne das Gesamtsystem wesentlich zu beeinflussen.

3. Eine gute Skalierbarkeit bezüglich der Menge der Peers, der Nachrichten für die Aktualisierung des Systems bei dynamischen Vorgängen, der Nachrichten für Suchvorgänge und der zeitlichen Auswirkungen bei den Aktualisierungen.

4. Aus den untersuchten konkreten Architekturen ergab sich, dass vor allem strukturierte Systeme mit Indexierung fast alle diese Anforderungen unterstützen können. Somit kann Strukturierung als Eigenschaft ein Hinweis auf eine geeignete Architektur sein.

Allerdings liegt die von den untersuchten Overlaystrukturen beherrschte Netzwerkdynamik nicht in einem Bereich, der ein Ad-hoc-ähnliches Verhalten erwarten lässt. Für P-Grid, einem der wenigen vollständig implementierten, sehr gut dokumentierten und ausgereiften Systeme, liegen experimentelle Ergebnisse vor. Obwohl dieses Overlaynetzwerk explizit für dynamische Netzwerke konzipiert wurde, zeigen die Experimente, dass das P-Grid-Netzwerk mit ca. 300 Peers z.B. ca. 300 Minuten [ADHS05] auf Einschwingen braucht. Nach dem vollständigen Aufbau und Einschwingen benötigt die Beantwortung einer Suchanfrage durchschnittlich zwischen 6 und 15 Sekunden [ADHS05]. Unter diesen Bedingungen kann die Einbindung mobiler Endgeräte allein mit solch einem System daher nur eingeschränkt erfolgen.

4.3 Mobile Agentensysteme

Nach der Untersuchung von Overlaystrukturen aktueller Peer-to-Peer-Systeme im vorangegangenen Kapitel widmet sich dieses Kapitel den mobilen Agenten, mobilen Agentensystemen und den Overlaystrukturen konkreter mobiler Agentensysteme.

4.3.1 Mobile Agentensysteme und mobile Agenten

Mobile Agenten *(MA)* sind eine spezielle Form von mobilem Code und eine spezielle Form von Softwareagenten[2]. Sie enthalten immer Programmcode, Zustand und Daten [BR05]. Im Gegensatz zu anderen Architekturmodellen verteilter Systeme reisen MA in einem Netzwerk von Rechner zu Rechner, um eine bestimmte Aufgabe *(User Task)* für eine bestimmte Person oder Institution auszuführen. Dabei nutzen sie grundsätzlich die lokalen Ressourcen der besuchten Rechner *(Agentenplattformen)* und führen vor Ort Berechnungen auf lokalen Daten aus, deren Ergebnisse sie dann mit sich führen. Am Ende ihrer Reise kehren sie mit dem Endergebnis, wie in Abbildung 4.9 dargestellt, zu ihrem Eigentümer zurück.

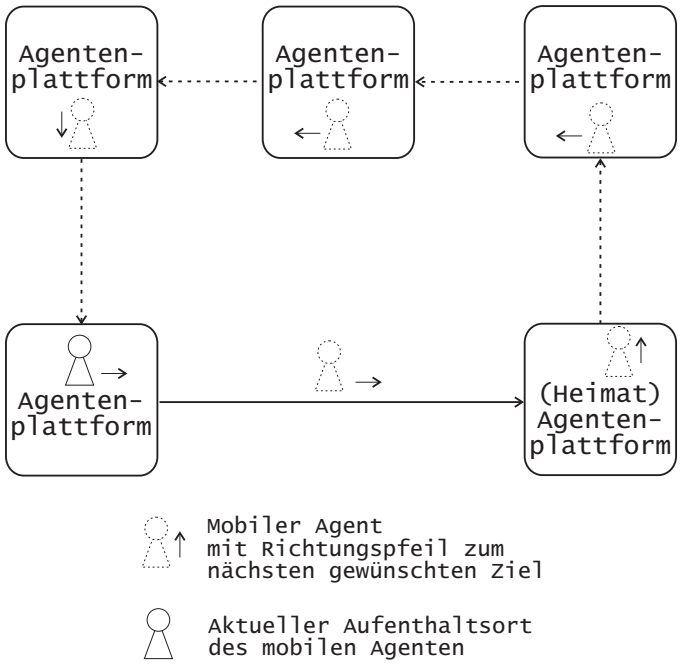

Abbildung 4.9: Interaktion im mobilen Agentensystem

Aber dies ist noch nicht alles: Die im ausführbaren Code von MA implementierte Aufgabe *(Task)* können sie in *autonomer* Weise erfüllen. Dabei nutzen sie die von ihrer aktuellen Agentenplattform zur Verfügung gestellten Ressourcen, rechnen also zwar zeitlich immer nur auf einer Plattform, aber in Erfüllung ihrer Gesamtaufgabe sequentiell verteilt. Mobile Agenten kommunizieren meist nur lokal auf der aktuellen Agentenplattform und nicht über das Netzwerk, so dass sie zum Teil von den Leistungsproblemen großer verteilter Systeme entkoppelt sind. Im Gegensatz zum Client-Server-Ansatz werden nicht die Daten

[2] Vgl. Kapitel 1, S. 4

zum Algorithmus, sondern der Algorithmus (in Form von Code), gekapselt im mobilen Agenten, zu den Daten transferiert. Bei der Wanderung *(Migration)* durch das Netzwerk von Agentenplattformen verwenden mobile Agenten verschiedene Migrationsstrategien, um sich den Netzwerkbedingungen anzupassen und die Netzwerkbelastung so gering wie möglich zu halten [Bra03]. Dies erreichen sie konzeptbedingt auch durch die Selektion der mitgeführten Daten, da sie eventuell nur bereits berechnete Ergebnisse mitführen müssen.

Gegenüber herkömmlichen Ansätzen verteilter Systeme können die Vorteile mobiler Agenten wie folgt zusammengefasst werden:

- Delegation von User Tasks an MA, welche diese autonom erledigen (Entlastung des Benutzers),
- Asynchrones Arbeiten im Netzwerk, dadurch relative Unabhängigkeit von der Heimatplattform (wichtig für mobile Endgeräte mit wenig Rechenleistung und schwankender Netzwerkanbindung),
- Reduzierung der Netzwerklast durch Verteilung der Datenströme (Dezentralisierung).

Mobile Agentensysteme eines gewissen evolutionären Entwicklungsstadiums erlauben die Entkopplung der Anwendung und des Nutzers von der Auswahl der Ressourcen im verteilten System, indem sie mobilen Agenten die selbständige Suche und Auswahl von Ressourcen entsprechend ihres Auftrages ermöglichen. Darauf wird im Abschnitt 4.3.3 näher eingegangen.

4.3.2 Mobile Agentensysteme als verteilte Systeme

Zum Besuch eines Rechners benötigen mobile Agenten eine spezielle Software, welche die Ausführungsumgebung für MA darstellt und als *Agency* oder auch *Agentenserver* bezeichnet wird. Computer im Netzwerk (auch *(Host)* genannt), die einen Agentenserver tragen und ausführen, werden als *Agentenplattformen* bezeichnet. Agentenplattformen stellen MA die benötigten Ressourcen in Form von Speicherplatz, Prozessorzeit und Diensten, z.B. Kommunikations- und Ressourcenverzeichnisdienste, zur Verfügung. Die logisch vernetzten Agentenplattformen bilden ein verteiltes System, ein *mobiles Agentensystem* [Bra03, BR05] bzw. *mobiles Agentennetzwerk*[3] [Erf04].

MA nutzen das mobile Agentensystem als Infrastruktur: Sie nehmen die von den Agentenplattformen angebotenen Dienste in Anspruch. Aus ihrer Sicht stellen die Agentenplattformen somit Server dar, während sie selbst dienstnutzende Clients sind.

[3] Anmerkung des Autors: Der Begriff mobiles Agentennetzwerk ist an dieser Stelle irreführend, da nicht die mobilen Agenten, sondern die Agentenplattformen vernetzt werden.

Aus Sicht der verteilten Systeme kann man die vernetzten Agentenplattformen als verteilte Komponenten ansehen, welche Dienste im Netzwerk anbieten und mit Hilfe mobiler Agenten asynchron miteinander kommunizieren. Durch simultanes Dienstanbieten und Dienstinanspruchnehmen agieren Agentenplattformen in der Rolle des Peers im klassischen Peer-to-Peer-System.

Aus Sicht einer verteilten Anwendung kann ein mobiles Agentensystem je nach Integrationsgrad und Fähigkeiten verschieden wahrgenommen werden, wie Abbildung 4.10 zeigt.

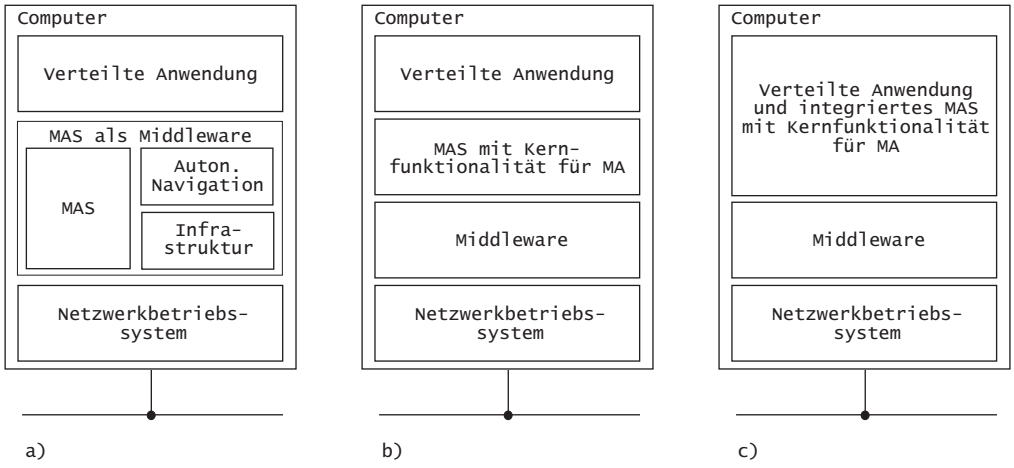

Abbildung 4.10: Mögliche Funktionsintegrationen in mobile Agentensysteme aus Sicht verteilter Anwendungen

Eine verteilte Anwendung kann ein mobiles Agentensystem als Middleware wahrnehmen (Abbildung 4.10 Bild a), wenn dieses Dienste zur autonomen Navigation und Routenfindung (wird im Abschnitt 4.3.3 eingehender behandelt) unterstützt. Durch diese Dienste muss sich die verteilte Anwendung nicht um die Ressourcenlokation kümmern, sondern kann dies dem MA und dem MAS überlassen. Das Ansprechen entfernter Ressourcen erfolgt dann autonom durch die Migration des MA, wodurch für die verteilte Anwendung Ortstransparenz bei der Inanspruchnahme der genutzten Ressourcen geschaffen wird.

Anderseits kann auch ein mobiles Agentensystem nur spezielle Dienste für mobile Agenten anbieten und ihrerseits eine andere Middleware nutzen (wie im MASIF-Standard vorgeschlagen, siehe Abschnitt 4.3.5.1), um selbst eine Zugriffstransparenz auf entfernte Ressourcen zu erhalten. In diesem Falle stellt das mobile Agentensystem, wie in Abbildung 4.10 Bild b dargestellt, eine Anwendungsschicht oberhalb der Middleware und unterhalb der eigentlichen verteilten Anwendung dar.

Ferner kann auch die verteilte Anwendung selbst in das mobile Agentensystem

integriert sein (oder anders herum), so dass sich ein spezielles, anwendungsbezogenes Agentensystem ergibt. Abbildung 4.10 Bild c zeigt die logischen Funktionsschichten, die in ein MAS integriert werden können.

4.3.3 Eine mögliche Taxonomie für mobile Agentensysteme

Mobile Agentensysteme sind eine relativ junge Erscheinung im Gebiet der verteilten Systeme. Nach ersten Erfahrungen mit realen Implementierungen um die Jahrtausendwende zeichnet sich nun, zumindest in der Forschung, ein qualitativer Sprung in den Fähigkeiten mobiler Agentensysteme ab. Erfurth [Erf04] charakterisiert aus der Sicht des Anwenders drei Stufen von Agentensystemen, welche sich in erster Linie durch den Aufwand und die Komplexität bei der Programmierung der mobilen Agenten unterscheiden.

Bei einem mobilen Agentensystem der *Stufe 1* muss für jede Aufgabe explizit ein spezieller Agent programmiert werden, wobei der Grad seiner Intelligenz (und damit der Programmieraufwand) von der übertragenen Aufgabe abhängt und vom Programmierer implementiert werden muss. Ferner muss dem mobilen Agenten die Reiseroute, also die zu besuchenden Zielpunkte und deren Reihenfolge, explizit übergeben werden. Daraus ergibt sich zwangsläufig eine statische Route, die vor Beginn der Reise des mobilen Agenten geplant werden muss. Auf Veränderungen der Umgebungssituation kann der Agent nicht reagieren. Diese Lösung eignet sich daher nur für ein statisches Netzwerk und somit nicht für die Einbeziehung dynamischer Ressourcen und mobiler Endgeräte.

Ein mobiles Agentensystem der *Stufe 2* stellt eine evolutionäre Weiterentwicklung des Systems der Stufe 1 dar und soll den Programmierer bei der Programmierung des Agenten von dessen Zielsuche und Routenplanung entkoppeln. Der Programmierer kann sich auf die Programmierung des Anwendungsalgorithmus konzentrieren (auf dass, *WAS* der Agent tun soll) und kann die Suche nach Plattformen mit geeigneten Diensten (Ziele) und die Reihenfolge (Route) des Besuchs (das *WIE* bzw. *WO*) den Agentenplattformen des Agentensystems überlassen. In dieser Stufe von Agentensystemen sind mobile Agenten wesentlich autonomer als in Systemen der Stufe 1, da sie ihre Zielplattformen selbst suchen und ihre Route selbst planen und optimieren lassen können. Sie können natürlich ihre Route auch ständig überarbeiten lassen, wenn sie dies für vorteilhaft ansehen (z.B. nach einer bestimmten Anzahl von Migrationen oder nach einer gewissen Zeit, wenn eine Veränderung in der Umgebung zu erwarten ist). Als Voraussetzung für den beschriebenen Schritt zum Agentensystem der Stufe 2 müssen deshalb folgende Dienste für mobile Agenten angesehen werden:

- ein Netzwerkerkundungsdienst (z.B. QoS-Daten auf logischer Netzwerkebene [EDR04]),
- ein Umgebungsdienst (z.B. in Form einer Karte *(Map)*),

- ein Routenplanerdienst (der auf der Umgebung/den Kartendaten arbeitet) und ein
- Routenoptimierer (optional, kann mit dem Routenplaner kombiniert werden).

Erfurth [Erf04] beschreibt in seiner Dissertation u.a. eine Implementierung dieser Funktionsumfänge und deren Eigenschaften in Form verschiedener Dienste in seinem Rahmenwerk *ProNav*, welches auf das mobile Agentensystem TRACY [B+03] der Friedrich-Schiller-Universität Jena aufbaut und dessen Funktionsumfang zu einem Agentensystem der Stufe 2 erweitert.

Als Basis für ProNav dient wiederum ein Dienst namens *Domain Service*, welcher die automatische Vernetzung der Agentenplattformen zu einem Agentensystem sicherstellt. Der Domain Service ist also ein Infrastrukturdienst, der eine Overlaystruktur zur Verwaltung von Diensten und Plattformen erzeugt, auf deren genaue Eigenschaften im Abschnitt 4.3.6.2 eingegangen wird.

Das Ziel eines mobilen Agentensystems der *Stufe 3* ist die Entkopplung des Benutzers von der Programmierung eines Anwendungsalgorithmus. Es bietet dann lernfähige Standardagenten an bzw. stellt Bausteine zum adaptiven Zusammenbauen solcher in Aussicht. An dieser Stelle kann die Programmierung eines Agenten mit Hilfe eines Avatars oder anderer intelligenter Möglichkeiten der Benutzerführung durch eine simple Abfragesequenz maskiert und von einem normalen Computernutzer erledigt werden. Der Eingriff eines Programmierers (eines Spezialisten) soll dann nicht mehr nötig sein.

4.3.4 Anforderungen an Infrastrukturdienste für MAS der Stufe 2

Mobile Agentensysteme der Stufe 2 benötigen zur autonomen Routenplanung und Optimierung aktuelle Dienstinformationen. Damit die Routenplanung effizient vonstatten gehen kann, sollten die Dienstinformationen schon vor der eigentlichen Planungsphase auf der Agentenplattform vorliegen. MA der Stufe 2 werden jedoch so proaktiv und autonom agieren, dass ihr Verhalten nicht mehr oder nur sehr begrenzt vorhersagbar sein wird [Erf04]. Somit kann ein dienstverwaltender Infrastrukturdienst eines MAS im vorhinein nicht wissen, welche Dienste ein beliebiger MA auf der Agentenplattform demnächst nachfragt. Es muss folglich alle möglichen Dienstinformationen (speziell von den Diensten, die ein MA zur Erledigung seines User Tasks sucht) sammeln und zur Verfügung stellen. Dies bedeutet zum einen, dass das Sammeln von Dienstinformationen permanent vonstatten gehen muss und nicht, wie in anderen Peer-to-Peer-Systemen, ausschließlich eventgetrieben erfolgen kann. Zum anderen skaliert das Sammeln sämtlicher potentiell im Internet verfügbaren Dienstinformationen nicht, da mengenmäßig weder die schiere Anzahl der Dienste noch die benötigte Anzahl von Nachrichten

zum Sammeln und Warten der Dienstinformationen in einem dynamischen Netzwerk beherrscht werden könnten. Trotzdem soll ein Infrastrukturdienst für MAS der Stufe 2 beide, zum Teil einander entgegenstehende Anforderungen erfüllen.

MAS der Stufe 2 unterstellen explizit dynamische Netzwerke. Erfurth wies in seiner Dissertation [Erf04] nach, dass mit den bisherigen Ansätzen eine Netzwerkdynamik von Diensten beherrscht werden kann, bei der ca. 60 Peers in 10 Minuten aktualisiert werden können [Erf04]. Mobile Agentenplattformen können sich aber für wesentlich weniger Zeit in einem MAS aufhalten. Als Anwendungsszenario soll hierbei ein MA dienen, der sich auf einer mobilen Agentenplattform befindet, die mangels Netzwerkverbindung in kein MAS eingebunden ist. Der Plattform liegen somit keine aktuellen Dienstinformationen vor und der MA kann sich keine Route planen lassen. Tritt die zwischenzeitlich bewegte Agentenplattform in ein Netzwerk ein und bekommt Kontakt zu einem MAS, kann der MA, wenn die Agentenplattform einen schnellen Zugriff auf aktuelle Dienstinformationen bekommt, sich umgehend eine Route planen lassen und seine Migration beginnen. Liegt der Aufenthalt der Agentenplattform im Netzwerk nur im Bereich weniger Sekunden, sollte dem MA wenigstens die Migration zu einer anderen Agentenplattform ermöglicht werden, die wesentlich mehr Potential zur Routenplanung (eine stabilere Umgebung) bietet als seine Heimatplattform. Ein Infrastrukturdienst für ein MAS der Stufe 2 sollte daher auch die Einbindung von hochdynamischen mobilen Agentenplattformen, die nur wenige Sekunden im MAS verweilen, unterstützen.

Im folgenden Abschnitt 4.3.5 werden die wichtigsten Standards im Bereich mobiler Agentensysteme und im darauf folgenden Abschnitt 4.3.6 konkrete MAS bezüglich der Eigenschaften ihrer Infrastrukturdienste und der hier aufgestellten Anforderungen untersucht und Defizite aufgezeigt. Ausgehend von den in diesem Abschnitt aufgestellten Anforderungen wird im nächsten Kapitel 5 ein Lösungsvorschlag für ein Framework aus Infrastrukturdiensten vorgestellt.

4.3.5 Standards für Agentensysteme

Im Bereich der Agententechnologie existieren zur Zeit zwei bedeutende Standardisierungsgremien, die *Foundation for Intelligent Physical Agents* (FIPA) und die *Object Management Group* (OMG), welche beide eigene Standards für Agentensysteme launchierten. Die von der FIPA herausgegebenen FIPA-Standards definieren abstrakte Standards für die gesamte Infrastruktur beliebiger agentenbasierter Anwendungen. Der MASIF-Standard der OMG ist hingegen spezieller ausgelegt und behandelt die Interoperabilität zwischen verschiedenen Agentensystemen, insbesondere bei der Verwendung mobiler Agenten. Aufgrund der sich teilweise überlappenden Arbeitsgebiete koordinieren FIPA und OMG ihre Standardisierungsbemühungen seit 1999 [Fou99].

In diesem Kapitel wird untersucht, inwieweit die erwähnten Standards Frage-

stellungen der Infrastruktur mobiler Agentensysteme und der zugrundeliegenden Middleware behandeln. Ein besonderes Augenmerk liegt auf Empfehlungen für die Realisierung solcher Infrastrukturen und Middleware und auf den daraus folgenden Eigenschaften, gerade in Bezug auf dynamische Netzwerkumgebungen.

4.3.5.1 Die Infrastruktur im MASIF-Standard

Der aktuelle MASIF-Standard [The00] stammt vom Januar 2000 und hat die Interoperabilität von *mobilen Agentensystemen* und nicht etwa von *mobilen Agenten* zum Ziel. Mobile Agentensysteme kommunizieren über die Migration mobiler Agenten, woraus sich fast zwingend die Verwendung der selben Programmiersprache der beteiligten Agentensysteme ergibt. Interoperabilität verschiedener Programmiersprachen zu erreichen ist zwar möglich, aber um einiges komplizierter. So fordert auch der MASIF-Standard die Verwendung der gleichen Programmiersprache der beteiligten Agentensysteme, welche aber ansonsten verschieden sein können.

Standardisierte Bereiche im MASIF-Standard Der MASIF-Standard [The00] vom Jahr 2000 standardisiert folgende Gebiete:

Das Agentenmanagement *(agent management)* beinhaltet die Erzeugung und Beendigung von Agenten sowie das Aussetzen und Fortsetzen von Agententhreads. Es bildet die Standardoperationen auf Agenten innerhalb ihres Lebenszykluses ab.

Der Agententransfer *(agent tranfer)* definiert einen formalen Ablauf zum Transferieren von Agenten. Löst ein Agent eine Migrationsanforderung aus, übergibt er gleichzeitig Namen und Adresse der Zielplattform sowie eine geforderte Verbindungsqualität an. Hier ist interessant, dass der Standard diese Verbindungsqualität nicht näher spezifiziert, sondern die Spezifizierung und Realisierung komplett dem Agentensystemhersteller überlässt.

Die standardisierte Namenssyntax für mobile Agenten *(agent names)* und Agentensysteme *(agent system names)* ermöglicht ein gezieltes Identifizieren und Ansprechen von Agenten und Agentensystemen.

Die Typisierung verschiedener Agentensysteme *(agent system type)* mit einzigartigen Identifizierern ermöglicht eine schnelle Zuordnung von Agenten zu den sie unterstützenden Agentensystemen. Dazu muss auch die Syntax zum Auffinden von Agentensystemen *(location syntax)* definiert sein.

Grundlegende Begriffe, Terminologie und Basiskonzepte Der MASIF-Standard definiert die üblichen Begriffe im Kontext mobiler Agentensysteme wie Agent, Agentenstatus usw. und unterscheidet auch mobile und stationäre Agenten. Interessant für die Betrachtung der Infrastruktur sind folgende Begriffe, die kurz

erläutert und zum Verständnis der Infrastruktur nach dem MASIF-Standard gebraucht werden.

Die Agentenautorität *(agent authority)* identifiziert den Eigentümer eines Agenten, also eine Organisation oder eine Person. Eine Autorität muss natürlich auch authentifiziert werden.

Der Agentenname *(agent name)* setzt sich aus dem Namen seiner Autorität, eines Identifizierers und seines Agentensystemtyps zusammen, um eine eindeutige Zuordnung zwischen Autorität und Agent gewährleisten und einen global einzigartigen Agentennamen kreieren zu können.

Der Agentenort *(agent location)* beschreibt die Adresse eines Platzes auf dem Agentensystem, auf dem sich der Agent gerade aufhält. Er enthält den Namen des Agentensystems und des *default places*.

Ein Agentensystem *(agent system)* stellt eine Plattform zum Erzeugen, Ausführen, Transferieren und Beenden eines Agenten dar. Dabei gehört das Agentensystem genau wie die Agenten einer bestimmten Agentenautorität an. Ein Hostrechner kann mehrere Agentensysteme beherbergen. Jedes davon ist durch seinen Namen und seine Adresse eindeutig identifizierbar.

Der Agentensystemtyp *(agent system type)* beschreibt den Hersteller, die Implementierungssprache und die Serialisierungsmechanismen einer bekannten Art von Agentensystem über einen Namen, z.B. “Aglet“ oder “Tracy“. Mit dem Agentensystemtyp seines Agentensystems wird auch das Profil eines Agenten *(agent profil)* festgelegt.

Ein Platz (oder Ort) *(place)* stellt eine Ausführungsumgebung für Agenten dar, welche innerhalb eines Agentensystems existiert. Damit ist es möglich, innerhalb eines Agentensystems mehrere Ausführungsumgebungen oder Ausführungskontexte zu schaffen, aber nicht zwingend nötig. Im letzteren Fall ist die einzige Ausführungsumgebung des Agentensystems als *(default place)* definiert.

Als Region *(region)* wird eine Menge von Agentensystemen der gleichen Autorität bezeichnet, welche aus verschiedenen Agentensystemen bestehen kann.

Der Agentenort *(agent location)* beschreibt die Adresse eines Platzes auf dem Agentensystem, auf dem sich der Agent gerade aufhält. Er enthält den Namen des Agentensystems und des default places.

Die Kommunikationsinfrastruktur *(communications infrastructure)* bietet einem Agentensystem einen Kommunikations-, Namens- und Sicherheitsdienst. Beim MASIF-Standard basiert dieser Dienst auf CORBA-Diensten.

Infrastrukturelemente und deren Zusammenspiel Zum Aufbau einer Infrastruktur nach dem MASIF-Standard kann man die Strukturelemente Platz, Agentensystem und Region benutzen (siehe Abbildung 4.11). Während die Strukturelemente Platz und Agentensystem technischer und administrativer Natur sind und sich explizit auf einem Hostrechner befinden müssen, ist die Region ein rein

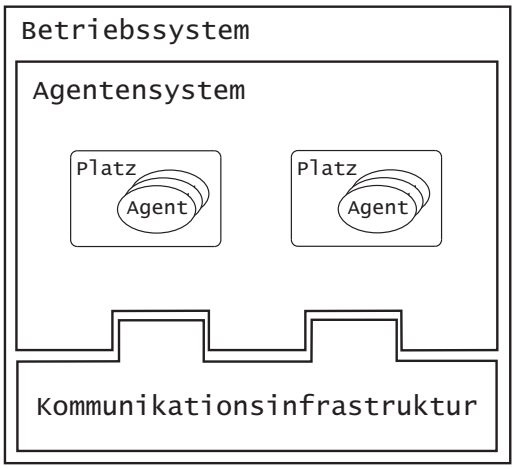

Abbildung 4.11: Aufbau eines MASIF-konformen Agentensystems

administratives Strukturelement. Sie beschreibt eine logisch zusammengehörige Domäne von Agentensystemen gleicher Autorität, die nicht zwingend an technische Grenzen der unterliegenden Netzwerkinfrastruktur, wie Switches, Router, IP-Bereiche u.ä., gebunden ist und sich über Netzwerkgrenzen hinweg erstrecken kann.

Innerhalb einer Region bestehen potentiell zwischen allen Agentensystemen Punkt-zu-Punkt-Verbindungen, so dass ein logisches, vollständig vermaschtes Netzwerk entsteht. Zum Auffinden von sämtlichen vorkommenden Entitäten wird ein zentraler Namens- und Verzeichnisdienst *(MAFFinder)* geführt, in dem alle Agenten, Places und Agentensysteme einer Region in jeweils einer Liste registriert sind. Die genannten Entitäten müssen sich explizit beim *MAFFinder* registrieren, um in die jeweilige zugeordnete Namensliste aufgenommen zu werden. Beim Verlassen der Region ist explizit eine Deregistrierung nötig. *MAFFinder* kann auch auf mehr als eine Region ausgeweitet werden, wobei aber dieses Szenario im MASIF-Standard nur genannt und nicht weiter betrachtet wird.

Für die Kommunikation nach außerhalb braucht jede Region mindestens einen Regionzutrittspunkt *region access point*, also eine oder mehrere Agentensysteme, welche nach außen sichtbar sind und eine Art Gateway mit Firewall auf Agentenebene darstellen. In Abbildung 4.12 sind die Infrastruktur nach dem MASIF-Standard und mögliche Kommunikationskanäle dargestellt.

Es sei noch angemerkt, dass die Infrastruktur im MASIF-Standard komplett auf der Verwendung von Diensten einer unterliegenden Middleware beruht, welche durch die ebenfalls von der OMG standardisierten CORBA-Plattform [The05] realisiert ist.

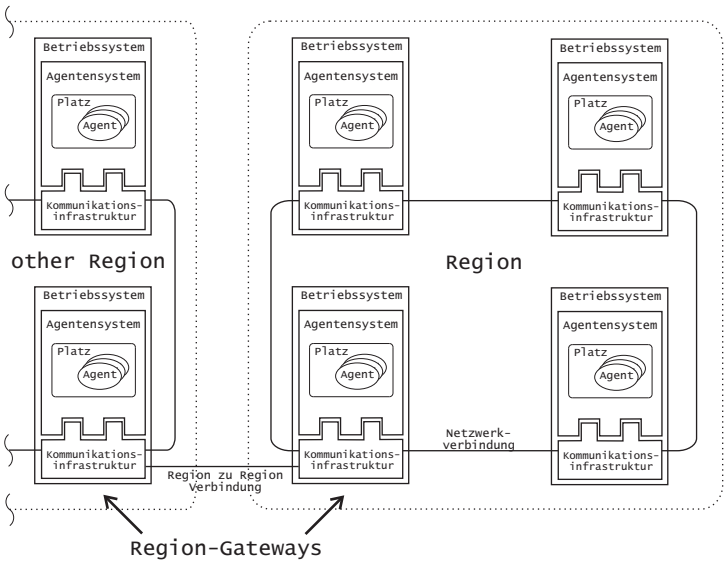

Abbildung 4.12: MASIF-Infrastrukturelemente und deren Zusammenspiel

Bewertung MASIF Der MASIF-Standard ist heute der wichtigste im Bereich der mobilen Agentensysteme und hat durch eine gewisse Verbreitung Bedeutung erlangt. Verschiedene kommerzielle Produkte, wie z.B. *Grasshopper* bzw. sein Nachfolger *enago mobile* [TA04], und nichtkommerzielle Forschungsprojekte, wie u.a. *SOMA* [Bol05] und *Aglets* [Agl04], unterstützen ihn heute teilweise oder ganz.

In Bezug auf eine unter der logischen Schicht der Agentensysteme liegenden Infrastrukturschicht, welche die logische Vernetzung und Strukturierung der Agentenplattformen sowie deren Zutritt und Austritt aus dem Netzwerkverbund verwaltet, bietet MASIF weder Normen noch Vorschläge. Er betrachtet dieses Aufgabenfeld überhaupt nicht und überlässt es dem Agentensystemhersteller.

Der Ansatz einer Region als logische Zusammenfassung mehrerer, eventuell verschiedener Agentensysteme auf Basis ihrer gleichen Autorität schafft hierbei eine weitere logische Betrachtungsebene oberhalb der Agentensysteme. Daher ist dieses Konzept zusätzlich in ein bestehendes Agentensystem integrierbar.

Der MASIF-Standard scheint sich auf eine fixe, nichtdynamische Umgebung zu beziehen, denn Maßnahmen zum Abfangen von Skalierungsproblemen und einer hohen Dynamik der beteiligten Netzwerkinstanzen wurden nicht getroffen.

Durch die unstrukturierte Vernetzung von Agentensystemen innerhalb einer Region kann dieser Regionen-Ansatz nicht mit der Menge der potentiell möglichen Anzahl von beitrittswilligen Agentensystemen skalieren. Auch die MAF-Finder-Verzeichnislisten, in denen jeder Agent, jeder Platz und jedes Agentensystem eingetragen sind, skalieren nicht.

4.3.5.2 Die Infrastruktur im FIPA-Standard

Im Fokus des FIPA-Standards [Fou07] standen von Anfang an *intelligente Agenten*, welche an sich stationär sind. Durch die fehlende Mobilität ergibt sich zwingend auch ein anderes Kommunikationsparadigma. So kommunizieren intelligente Agenten i.d.R. über Nachrichten in einer Agentenkommunikationssprache *(Agent Communication Language, ACL)*, z.B. KQML/KIF [FFMM94] oder FIPA-ACL/SL [Fou02b], welche der Sprechakttheorie *(speech-act-theory)* folgen. Agentensprachen auf diesem relativ hohen Abstraktionsniveau ermöglichen auch die Kooperation von verschiedenen Agenten und Agententypen, welche sogar unterschiedlichen Agentensystemtypen angehören dürfen. In Verbindung mit der ACL werden Ontologien verwendet, welche für jeweils einen Anwendungsbereich gültige Mengen von Fähigkeiten darstellen und quasi einer API auf RPC-Ebene gleichen.

Der FIPA-Standard versucht somit Interoperabilität auf Kommunikationssprachenbasis herzustellen, also auch zwischen Agentensystemen, welche nicht in der gleichen Sprache geschrieben sind.

Standardisierte Bereiche im FIPA-Standard Mit dem FIPA-Standard wird versucht, eine alles überspannende, offene Infrastruktur für Agentensysteme zu schaffen. Die hier gemeinte Infrastruktur bezieht sich allerdings nur auf die Agentenebene selbst. Die zu standardisierenden Bereiche sind entsprechend der Thematik in einzelne Spezifikationen aufgeteilt. Dabei umfasst die *FIPA Abstract Architecture* [Fou02a] alle Bereiche, die für einen semantisch bedeutungsvollen Nachrichtenaustausch notwendig sind. Dazu gehören u.a. auch die wichtigsten Infrastrukturelemente, von denen die für diese Dissertation relevanten im nächsten Abschnitt vorgestellt werden. Ferner existieren u.a. noch anwendungsbezogene Spezifikationen, welche spezielle Sprechakte und Ontologien festlegen. Dazu gehört z.B. die *FIPA Quality of Service Ontology Specification* [Fou02c] für den Austausch von Nachrichten für auf QoS-Informationen angewiesene Agenten und Anwendungen. Die *FIPA Policies and Domains Specification* [Fou01b] hingegen definiert administrative Architekturelemente zur Unterstützung von Policies.

Grundlegende Begriffe, Terminologie und Basiskonzepte Im Rahmenwerk des FIPA-Standards [Fou02a] werden folgende Entitäten und Begriffe zur Infrastrukturbildung benutzt:

Eine Agentenplattform *(agent-platform)* stellt eine Ausführungsumgebung für Agenten dar.

Als Agent *(agent)* wird ein Prozess bezeichnet, der eine autonome (entscheidungsautonome), kommunizierende Funktionalität einer Anwendung erfüllt.

Er hat einen Agentennamen *(agent-name)*, der ihn eindeutig identifiziert und

kann Attribute *(agent-attributes)* besitzen, welche seine Eigenschaften näher beschreiben.

Der Agentenort *(agent-locator)* beschreibt seine Kommunikationsendpunkte *(trans-port-descriptions)*, welche wiederum ein Tupel aus dem verwendeten Transportprotokoll *(transport-type)*, der protokolltypischen Adresse des Endpunktes *(transport-specific-address)* und optionalen transportprotokolltypischen Argumenten *(transport-specific-properties)* besteht.

Ein Agent kann sich mit Hilfe seines Eintrages im Agentenverzeichnis *(agent-directory-entry)* an einem oder mehreren Agentenverzeichnisdiensten *(agent-directory-services)* registrieren, um sich anderen Agenten bekannt zu machen. Der Eintrag enthält neben seinem Namen auch seinen Agentenort. Das Durchsuchen der Agentenverzeichnisse ermöglicht dem Agenten das Finden anderer Agenten, während ihm die im jeweiligen Agentenverzeichniseintrag vorhandenen Agentenorte die Kommunikation *(transport-message)* ermöglichen. Ein Agent kann seinen Eintrag jederzeit verändern oder sich deregistrieren.

Um seine spezifischen Aufgaben erfüllen zu können, muss ein Agent nicht nur mit anderen Agenten kommunizieren, sondern auch Dienste *(Services)* in Anspruch nehmen können. Obwohl Dienste auch von Agenten angeboten werden können, wird im FIPA-Standard an dieser Stelle insoweit zwischen Diensten und Agenten unterschieden, indem für Dienste ein eigener Verzeichnisdienst *(Service-directory-service)* aufgebaut wird. So erzielt man durch die Entkopplung von Agenten und Diensten Vorteile: Dienstverzeichnisse und Dienste können auch von anderen Softwareentitäten als Agenten genutzt bzw. in Anspruch genommen werden. Die FIPA-Infrastruktur kann dadurch auch für normale, agentenunabhängige Dienste wie WebServices verwendet werden, z.B. zur Aggregation von Grunddiensten zu komplexeren, spezielleren Diensten.

Um die Entität des Dienstes herum definiert der FIPA-Standard eine ähnliche Umgebung wie um den Agenten. Der Dienst hat einen einzigartigen Dienstnamen *(Service-name)* und einen Dienstort *(Service-locator)*, bestehend aus einer Menge von Dienstortbeschreibungen *(Service-location-description)*, welche wiederum ein Tupel aus dem Dienstsignaturentyp *(Signature-type)*, der Dienstsignatur *(Service-signature)* und der Dienstadresse *(Service-address)* ist. Im Dienstverzeichnis kann sich ein Dienst mit seinem Dienstverzeichniseintrag *(Service-directory-entry)* registrieren. Der Eintrag besteht aus dem Dienstnamen, dem Dienstort, dem Diensttyp *(Service-type)*, er kann zusätzlich noch Diensteigenschaften *(Service-attributes)* enthalten.

Zusätzlich zur *FIPA Abstract Architecture* werden im Kontext von Sicherheitsrichtlinien *(policies)* zur Steuerbarkeit von und Rechtezuweisung an Agenten in der *FIPA Policies and Domains Specification* administrative Architekturelemente definiert. Ein wichtiger Begriff in diesem Zusammenhang ist hier die Domäne *Policy Domain*, welche im FIPA-Standard meist nur kurz Domain genannt wird

und eine Menge von Agenten mit der gleicher Sicherheitsrichtlinie bezeichnet. Wird ein Agent auf einer Agentenplattform erfolgreich gestartet, erhält er automatisch die Sicherheitsrichtlinie der Agentenplattform. Domänen sind weder auf eine Agentenplattform noch auf einen Computer *(Host)* begrenzt. Mit der Fähigkeit, mehrere Agentenplattformen auf einem Host zu starten, ergibt sich auch die Möglichkeit, dass ein Host mit mehreren Agentenplattformen mehreren Domänen angehört.

Ein *Domain Manager* ist eine in einer Domäne einzigartige, zentrale Instanz zur Verwaltung der Sicherheitsrichtlinien. An ihm muss jeder der Domäne angehörende Agent registriert sein.

Die Verwendung von bereichsspezifischen Ontologien als gemeinsame Begriffswelt zweier miteinander kommunizierender Agenten wurde schon im vorherigen Abschnitt erwähnt. Eine Ontologie ist im Zusammenhang mit der Infrastruktur interessant: Die *FIPA Quality of Service Ontology Specification* [Fou02c], welche zum Austausch von QoS-Daten in entsprechenden Anwendungen dient. Auf die Einzelheiten dieser Spezifikation soll hier nicht weiter eingegangen werden, es sei aber angemerkt, dass die Existenz einer solchen Ontologie natürlich die Möglichkeit bietet, diese als Nachrichtenformat in eigenen Anwendungen zu verwenden.

Infrastrukturelemente und deren Zusammenspiel Im FIPA-Standard besteht die Infrastruktur aus Agentenplattformen, auf denen die Agenten ausgeführt werden. Es können mehrere Agentenplattformen auf einem Host residieren. Agentenplattformen unterliegen Sicherheitsrichtlinien, welche auch für die auf ihr ausgeführten Agenten Gültigkeit besitzt. Alle Agenten gleicher Sicherheitsrichtlinien bilden eine Domäne, also eine logische Zusammenfassung rechtlich gleichmächtiger Agenten über Agentensystem- und Hostgrenzen hinweg. Die Domäne wird vom Domänenmanager verwaltet.

Um andere Agenten zu finden, nehmen Agenten Agentenverzeichnisdienste in Anspruch. In den Agentenverzeichnissen sind alle kommunikationswilligen Agenten registriert. Agenten benötigen zur Abarbeitung ihrer Aufgaben ebenfalls Dienste, welche im Dienstverzeichnis des Dienstverzeichnisdienstes registriert sind.

Bewertung FIPA Der FIPA-Standard (es ist vielmehr ein Sammelsurium von locker zusammenhängenden Spezifikationen) ist der wichtigste Standard für Agenten und Agentensysteme im Allgemeinen. Er ist mittlerweile sehr mächtig und vielschichtig geworden, vor allem, seit FIPA und OMG zusammenarbeiten und immer mehr Einflüsse aus dem MASIF-Standard in die FIPA-Spezifikationen einfließen. Dies zeigt sich z.B. auch an der zunehmenden Berücksichtigung mobiler Agenten. Aktuelle Agentensysteme sind heute FIPA-konform, so z.B. *enago mobile* [TA04], das auch MASIF-konform ist, und die Multi-Agentenplattform

JADE [BCPR03], ein Produkt des Telecom Italia Lab.

Genau wie der MASIF-Standard gibt es im FIPA-Standard keinen Hinweis auf die intelligente Verwaltung der Hosts auf einer virtuellen Netzwerkschicht unterhalb der Agentensystemebene. Deshalb treten konzeptbedingt die gleichen Probleme auf: Der Zutritt und Austritt der Hosts aus dem Netzwerkverbund wird nicht verwaltet und damit nicht oder erst spät bemerkt, mobile Hosts inklusive ihrer Agentenplattformen und Agenten können einfach aus dem Verbund verschwinden und es wurden statische und begrenzte Netzwerke zugrunde gelegt, denn Maßnahmen zum Abfangen von Skalierungsproblemen und von hoher Dynamik der beteiligten Hosts wurden nicht getroffen.

4.3.5.3 MASIF und FIPA - Fazit

Es existieren viele Vergleiche [Lai01, PPW02, Mag05] der beiden Standards bezüglich der Kompatibilität und Harmonisierung. Dabei fallen vor allem die Ähnlichkeiten [Mag05] der Infrastrukturelemente *FIPA Domain* und *MASIF Region* sowie *FIPA Agent Platform* und *MASIF Agent System* auf. Die in dieser Dissertation untersuchten Infrastrukturschichten unterhalb der Agentensystemebene werden aber von keinem der beiden Standards betrachtet oder auch nur tangiert. Beide Standards unterstellen ein statisches Netzwerkszenario mit festen Hosts und festen Dienstzugangspunkten. Es wurden keinerlei Maßnahmen zur Skalierbarkeit und zur Beherrschung der dynamischen Netzwerkumgebung getroffen.

4.3.6 Konkrete mobile Agentensysteme

4.3.6.1 SOMA

Das *Secure and Open Mobile Agent (SOMA)* Environment der Universität Bologna stellt ein Rahmenwerk für die Programmierung von Anwendungen auf Basis mobiler Agenten dar. Dabei wurde bei SOMA der Fokus auf Sicherheit und Interoperabilität gelegt. So unterstützt es den Standard für mobile Agenten MASIF (siehe Abschnitt 4.3.5.1) sowie die Anbindung an CORBA-basierte verteilte Systeme [BCCS99]. Auf der Basis von SOMA wurden einige wissenschaftlich motivierte Anwendungen programmiert, allerdings fast ausschließlich von Angehörigen der Universität Bologna.

SOMAs Infrastrukturelemente *place*, *domain* und *protection locality* bilden ein logisches Netzwerk ab. Die *places* entsprechen den Agentenplattformen und stellen immer einen physischen Netzknoten dar. Mehrere *places* werden zu einer *domain* (Domäne) zusammengefasst. Die Domäne bezieht sich auf ein IP-Subnetzwerk, also auf ein Netzwerkstrukturelement. Ein spezieller *place* pro Domäne, der *default place*, fungiert als zentraler Gateway zu anderen Domänen.

Über diese Struktur hinweg können sich *protection localities* erstrecken, die logisch zusammengefasste Bereiche einer Administrationsautorität abbilden. Die Overlaystruktur SOMAs ist daher als hierarchisch und strukturiert einzustufen. Abbildung 4.13 zeigt die SOMA-Infrastruktur.

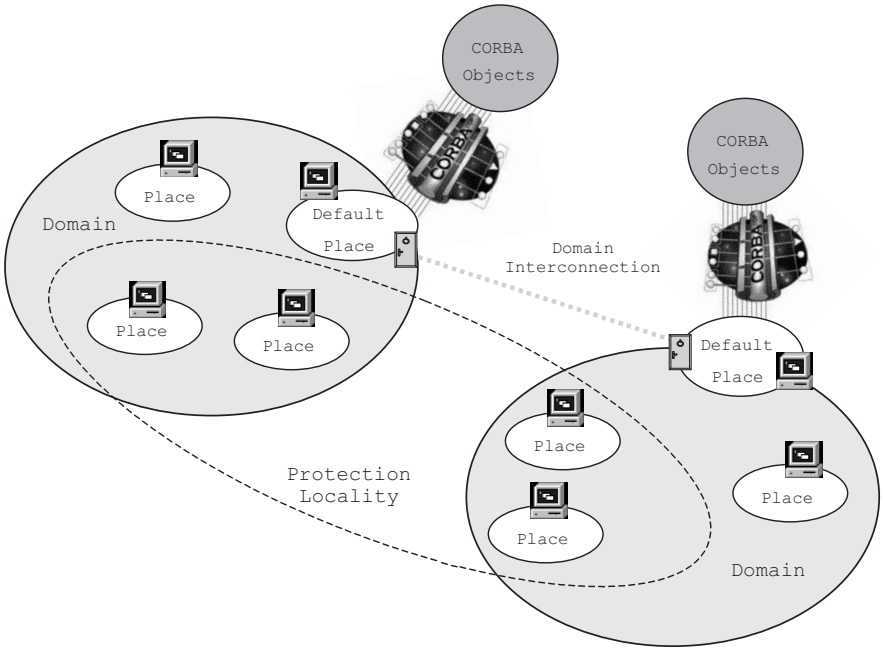

Abbildung 4.13: Die Infrastruktur von SOMA

Da sämtliche zentrale Funktionen auf dem *default place* realisiert werden, muss dieser Netzwerkknoten sehr zuverlässig sein. Bei SOMA wird deshalb von einem stabilen Computer mit stabiler und zuverlässiger Netzwerkanbindung ausgegangen. Damit bildet der *default place* ein festes Infrastrukturelement bzw. stellt einen festen Bezugspunkt der Infrastruktur dar. SOMAs Peers sind daher eng gekoppelt. Eine weitere Besonderheit und Abweichung vom reinen Peer-to-Peer-Ansatz stellt das Agentenrouting über die *default places* dar. Mobile Agenten, die in eine andere Domain migrieren wollen, müssen dies über die *default places* der eigenen und der Zieldomäne [Chi] tun. Neben dem Interdomänenrouting können die *default places* auch die Anbindung zu CORBA-basierten Nichtagentensystemen herstellen.

SOMA verwendet zur Identifizierung von Domänen, Agentenplattformen und mobilen Agenten global eindeutige Identifizierer *(globally unique identifier - GUID)*, die logische Namen darstellen und unabhängig vom momentanen Standort sind. Domänennamen identifizieren die Domänen, sind also an die Lokalität gebunden, und bezeichnen das Heimatnetzwerk einer Agentenplattform. Die Identifizierer der Agentenplattformen setzen sich aus dem heimatlichen Domä-

nennamen und einem im dortigen Netzwerk eindeutigen Hashwert zusammen. Ein Agentenname besteht wiederum aus dem Identifizierer seiner Heimatplattform versehen mit einem zusätzlichen Hashwert, der auf der Heimatplattform eindeutig ist. Sind die Namen vergeben, werden sie auch bei der Migration von Agenten und Agentenplattformen beibehalten. Ein Verzeichnisdienst übernimmt die Zuordnung von Identifizierer und aktuellem Aufenthaltsort im Netzwerk. Ein Vorteil dieser (oft verwendeten) Lösung ist die Erkennung der Heimatdomäne eines Agenten oder einer Agentenplattform anhand eines Teils seines Identifizierers in einer fremden Domäne, ohne einen Namensdienst bemühen zu müssen.

SOMA nutzt zur Bekanntmachung von Agentenplattformen in einer Domäne einen kombinierten Ansatz aus Discovery und Verzeichnisdienst. Ein ins Netzwerk eintretender Knoten initiiert einen Broadcast, der von einem Discovery-Server auf dem *default place* aufgefangen wird. Dieser Mechanismus wird gleichermaßen zum Registrieren wie zum Deregistrieren genutzt. Dabei wird von einer geringen Dynamik der Plattformen ausgegangen [BCS01], wofür dieser zentrale Ansatz ausreichend leistungsfähig sei. Auf einem ebenfalls auf dem default place installierten, LDAP-basierten Verzeichnisdienst können sich nun alle Ressourcen (Agentenplattformen, mobile Agenten, Dienste, u.a.), die global sichtbar sein wollen, registrieren.

Auf der Basis von SOMA wurden einige interessante Anwendungen im nichtkommerziellen Bereich realisiert, welche die Verbreitung und Reife des Systems unterstreichen. Einen Überblick dazu kann man [Bol05] entnehmen.

Bezogen auf die im Abschnitt 4.3.4 aufgestellten Anforderungen erfüllt SOMA nur wenige. So ist zumindest der Zugang einer Agentenplattform zum SOMA-Netzwerk ausreichend schnell realisiert. SOMAs Infrastruktur geht explizit von sehr geringer Netzwerkdynamik aus, was sich in der Art der zentralisierten Verwaltung ausdrückt. Über ein global vernetztes Gesamtsystem mit einer hohen Anzahl von Agentenplattformen werden keine Aussagen gemacht. Das bestehende SOMA-Konzept kann ohne Erweiterung große Systeme nicht unterstützen.

4.3.6.2 TRACY

An der Friedrich-Schiller-Universität Jena (FSU) wurde in den letzten Jahren am Lehrstuhl für Softwaretechnik das mobile Agentensystem TRACY [BER00, B+03] entwickelt und in Java2 implementiert.

Das Agentensystem Tracy ist ein Peer-to-Peer-System und besteht aus Agentenplattformen, welche Ausführungsplattformen für mobile Agenten darstellen. Die Agentenplattformen haben alle gleiche Fähigkeiten und den gleichen Funktionsumfang, stellen also Peers dar. Um die globale Menge der Agentenplattformen managen zu können, wird sie in disjunkte Mengen zerlegt und in einer Overlaystruktur organisiert. Den Aufbau der Infrastruktur übernimmt ein auf jeder Agentenplattform vorhandener Infrastrukturdienst namens TRACY *Domain*

Service. Zur Strukturierung der Infrastruktur stehen die Infrastrukturelemente *Domain Node*, *Domain* (Domäne), *neighborhood* (Nachbarschaft), *Domain Manager* und *Domain Master* zur Verfügung. Ein Domain Node ist eine normale Agentenplattform ohne spezifische, über den normalen Umfang hinaus gehende Infrastrukturdienste. Ein Domain Manager stellt eine spezialisierte Agentenplattform dar, welche die Verwaltung einer Domäne übernimmt und auch die Verbindung zum Domain Master hält. Der Domain Master entspricht wiederum einem spezialisierten Domain Manager, welcher für die Verbindung aller globalen Domänen verantwortlich und in seiner Aufgabe einzigartig ist. Das Tracy Overlaynetzwerk ist daher als strukturiert und streng hierarchisch organisiert einzustufen. Das zentrale Konzept des Domain Masters lässt bei begrenzter Peeranzahl eine gute Performance erwarten, skaliert aber konzeptbedingt im globalen Internet nicht. In Abbildung 4.14 ist eine TRACY Infrastruktur dargestellt.

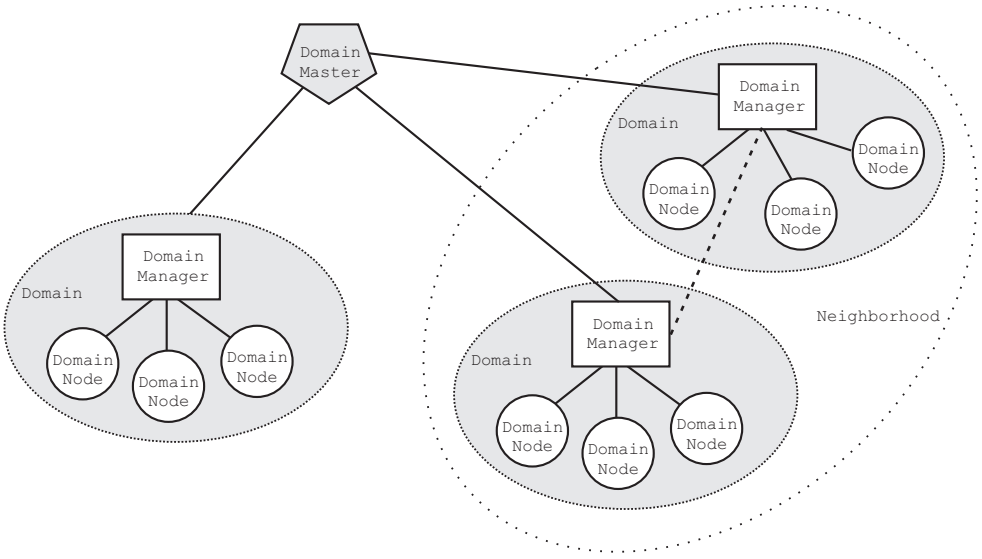

Abbildung 4.14: Der Infrastrukturaufbau von TRACY

Eine Besonderheit beim TRACY Domain Service ist der selbstorganisierende Ansatz zur dynamischen Bildung einer Domäne. Startet eine Agentenplattform ihren Domain Service in Form eines speziellen stationären Agenten, dem *Domain Information Agent*, so versucht dieser über einen lokalen, d.h. nur das eigene IP-Subnetz betreffenden, UDP-Broadcast einen vorhandenen Domain Manager zu erreichen. Antwortet der Domain Manager mit einem UDP-Datagramm, der seine URL beinhaltet, so registriert sich anschließend der sich anmeldende Domain Node am Domain Manager. Die Registrierung und sämtliche weitere Kommunikation erfolgt über den Austausch mobiler Agenten. In einem nichtbelasteten 100 Mbit/s Netzwerk dauert der Registrierungsvorgang durchschnittlich nur

40 ms [BER00], ist also sehr performant.

Antwortet kein Domain Manager innerhalb einer bestimmten Zeit, so nimmt der Domain Node, da er offensichtlich die erste Agentenplattform im IP-Subnetz ist, selbst die Rolle des Domain Managers an und startet spezielle andere Infrastrukturdienste, um seine Domäne zu verwalten. Danach meldet sich der Domain Manager beim Domain Master, der schon bekannt sein muss, an und gibt die Existenz seiner Domäne global bekannt. Er kann nun vom Domain Master eine Liste mit URLs von Domain Managern erhalten, die ihm dann direkt bekannt sind und seine Nachbarschaft darstellen. Der Ansatz der Nachbarschaft ist in TRACY bisher nicht weiter spezifiziert und vertieft worden. Ursprünglich sollte sich die Nachbarschaft einer Domäne auf andere, im Netzwerk gut erreichbare Domänen beziehen. Das Konzept der Nachbarschaft könnte aber prinzipiell auch genutzt werden, um logisch zusammenhängende Bereiche zu definieren, wie z.B. die im MASIF Standard spezifizierten Regionen, die Agentenplattformen der gleichen Autorität zu einer organisatorischen Einheit zusammenfassen.

Wird eine Agentenplattform beendet, so deregistriert sie sich beim Domain Manager, welcher sie aus der Liste seiner verwalteten Domain Nodes entfernt. Der Domain Manager verbreitet die aktuelle Domain Node Liste an alle seine Domain Nodes, sobald eine Änderung in der Domänenstruktur eingetreten ist. Er behält den Überblick über die Domain Nodes seine Domäne, indem er alle Domain Nodes mit einem Polling Mechanismus regelmäßig auf Anwesenheit überprüft. Fällt ein Domain Node aus, bemerkt dies der Domain Manager erst durch die ausbleibende Antwort der regelmäßig wiederholten Abfrage des fehlenden Domain Nodes. Mit zunehmender Anzahl der Domain Nodes in einer Domain vergrößert sich entweder das Abfrageintervall oder die Netzwerklast durch das Polling. Der Nachrichtenverkehr innerhalb einer Domain skaliert daher bei einer höheren Anzahl von Domain Nodes und dynamischem Verhalten nicht mehr.

Wird ein Domain Manager beendet, wählen die verbleibenden Domain Nodes einen neuen Domain Manager anhand des höchsten Prioritätswertes. Der Domain Node mit der höchsten Priorität wird Domain Manager. Der Prioritätswert ist ein in jede Agentenplattform fest einprogrammierter Wert und sollte der Leistungsfähigkeit des Hostrechners entsprechen. Die dynamische Wahl des Domain Managers wird auch ausgelöst, wenn ein neuer Domain Node mit höherem Prioritätswert als die des bestehenden Domain Managers in eine Domäne eintritt. Mit dieser Maßnahme wird die Kopplung der Domain Nodes an einen bestimmten Domain Manager verringert und auf dynamische Prozesse reagiert. Allerdings kann die Leistungsfähigkeit eines Domain Nodes zur Laufzeit schwanken, worauf nicht direkt eingegangen werden kann. Die implementierte Lösung mit zur Laufzeit fester Priorität unterstützt daher die Anpassung an dynamische Prozesse nur bedingt gut.

Agenten und Agentenplattformen werden in Tracy über ihre Namen angesprochen. Das Muster der Namensbildung ist dem von SOMA ähnlich. Dazu wird für jede Agentenplattform ein global eindeutiger Name erzeugt. Die Namen der Agenten werden vom Namen ihrer Heimatplattform, d.h. der Plattform, auf der sie erzeugt werden, abgeleitet. Dazu wird ein auf der Plattform eindeutiger Name für den Agenten gebildet und mit dem global eindeutigen Namen der Agentenplattform kombiniert. Da in TRACY Dienste nur über stationäre Agenten angeboten werden, können Dienste ganz einfach über die Namen ihrer stationären Agenten identifiziert und angesprochen werden.

Mittlerweile existiert eine neue Version von TRACY, TRACY2. Diese Software wurde von der Startup-Firma *The Agent Factory* (TAF), einer Ausgründung aus der FSU, aus TRACY heraus entwickelt. Dieses System besitzt eine völlig überarbeitete Architektur und erweiterte Funktionsumfänge. Am Domain Manager und der Verwaltung der Domänen hat sich allerdings nur die Kommunikationsart geändert, so dass für die Aufrechterhaltung der Domänenstruktur keine mobilen Agenten mehr verwendet werden.

Die im Abschnitt 4.3.4 aufgestellten Anforderungen erfüllt das TRACY Domain Konzept zum Teil. So ist für einen schnellen Eintritt einer Agentenplattform ins MAS und eine schnelle initiale Verteilung der Dienstinformationen gesorgt worden. Wie schon in Abschnitt 4.3.4 angesprochen, kann die Aktualisierung der Dienstverzeichnisse (Maps) laut Erfurth schon bei ca. 60 Peers etwa 10 Minuten dauern. Bei hochdynamischen Netzwerken kann es daher passieren, dass die Dienstinformationen nicht sehr aktuell sind. Dies wiederum kann zu fehlerhaften Routenplanungen führen und den Nutzen einer Routenoptimierung schmälern. Die Skalierung des Overlaynetzwerkes ist ebenfalls nicht sichergestellt.

Weiterführende Arbeiten in Bezug auf die Overlaystrukturierung von MAS wurden an der FSU am Lehrstuhl für Softwaretechnik vom Autor dieser Dissertation durchgeführt [DER05]. Ferner wurde am selben Lehrstuhl das Thema auch im Rahmen des Projektes *MobiSoft* [LSt07c] im anwendungsspezifischen Zusammenhang von Sozialen Netzwerken [KBD+06a] und von E-Business [KBD+06b], jeweils in Verbindung mit MANETs[4] aufgegriffen. Die vorgestellten Konzepte wurden aber so nicht weiter verfolgt. Im Umfeld Sozialer Netzwerke wurde jedoch ein Prototyp [KBR06] mit Hilfe von TRACY und JXTA implementiert.

4.3.6.3 Andere mobile Agentensysteme

Mobile Agentensysteme sind meist auf Basis von Forschungsarbeiten an verteilten Systemen an verschiedenen Universitäten heraus entstanden. Viele Projekte

[4] Die Abkürzung MANET steht für *Mobile Ad-hoc Network*, also Netzwerke, die ohne vorher installierte Infrastruktur auskommen. Vgl. [Ste04], S. 165/166.

in diesem Bereich werden heute nicht mehr aktiv verfolgt oder leben als Open Source Projekte fort. Die europäische Organisation und Interessengemeinschaft zu Agentensystemen *AgentLink* [ALk05] führt auf ihren Internetseiten eine Übersicht von ehemaligen und aktuellen Projekten.

Eine Betrachtung anderer mobiler Agentensysteme bezüglich der Bildung einer Infrastruktur in Form logischer Netzwerke kann man bei Erfurth [Erf04] nachlesen. Zusammenfassend betrachtet, existieren auf dem Gebiet der mobilen Agenten nur wenige Systeme mit Produktcharakter, wie z.B. *TRACY2 the agent factory GmbH* [Taf06] und *enago mobile* [TA04]. Die meisten anderen Systeme sind eher prototypisch implementiert oder haben Forschungscharakter. Die Organisation der Infrastruktur wird bei diesen Systemen meist nicht betrachtet oder nur sehr rudimentär gestaltet, da sie nicht als Kernfunktionaliät eines mobilen Agentensystems angesehen wird.

4.3.7 Zusammenfassung mobile Agentensysteme

Mobile Agenten handeln als autonome Einheiten in einem mobilen Agentensystem, bestehend aus Agentenplattformen. Sie migrieren von Plattform zu Plattform und nutzen dabei die verteilten Ressourcen der besuchten Plattformen. Mobile Agentensysteme der Stufe 2 stellen einen qualitativen Sprung in der Entwicklung dar, da der Programmierer bei der Programmierung des Agenten von dessen Zielsuche und Routenplanung entkoppelt wird. Somit wird aus Sicht des Programmierers die Verteilung der Ressourcen transparent, womit ein mobiles Agentensystem der Stufe 2 auch die Bedingungen einer Middleware vollständig erfüllt.

Mobile Agentensysteme der Stufe 2 stellen aber auch höhere Anforderungen an die Leistungsfähigkeit eines mobilen Agentensystems. Dies gilt insbesondere für die Infrastrukturdienste, welche sich um die Strukturierung des Agentensystems, die Einbindung neuer Plattformen und die Detektion und Propagation von Diensten kümmern. Soweit uns derzeit bekannt ist, erfüllt kein Infrastrukturdienst oder Infrastrukturdienst-Framework eines mobilen Agentensystems alle in Abschnitt 4.3.4 aufgestellten Anforderungen an Infrastrukturdienste für ein MAS der Stufe 2 gleichzeitig.

II

QuickLinkNet - ein Infrastrukturdienst-Framework für MAS der Stufe 2

5 Konzepte für Infrastrukturdienste für ein MAS der Stufe 2

Im Teil I dieser Arbeit wurden ausschnittsweise die Grundlagen der Kommunikationssysteme und der State-of-the-Art verteilter Systeme im Allgemeinen sowie der zu Peer-to-Peer-Systemen und Infrastrukturdiensten von MAS im Besonderen dargestellt. Es wurden ferner die Defizite von Infrastrukturen logischer Netzwerke bezüglich der Anforderungen eines MAS der Stufe 2 aufgezeigt.

Der Teil II dieser Arbeit widmet sich nun der Lösung der bisher aufgezeigten Problemstellung. Dazu wird zunächst in Kapitel 5 aus den Defiziten der untersuchten Infrastrukturen logischer Netzwerke die Eigenschaften eines Infrastrukturdienstes für ein MAS der Stufe 2 abgeleitet. Aus ihnen wird ein Lösungsansatz erarbeitet, der in einem allgemeinen Umsetzungs- und Architekturvorschlag mündet. Am Ende des Kapitels 5 werden aus dem Lösungsansatz die Thesen der Dissertation aufgestellt. Kapitel 6 widmet sich der konkreten Umsetzung der Thesen im Rahmenwerk *QuickLinkNet*, welches der vorgeschlagenen Architektur folgt und als komplexer Infrastrukturdienst die Anforderungen an ein MAS der Stufe 2 erfüllt. Der Aufbau und die Funktionsweise der einzelnen Bestandteile QuickLinkNets werden dann in den Kapiteln 7 bis 9 ausführlich beschrieben. Kapitel 10 fasst die Inhalte dieses Teils der Dissertation zusammen.

5.1 Anforderungen und gewünschte Systemeigenschaften

Im Abschnitt 4.3.4 wurden die Anforderungen an eine Infrastruktur für ein MAS der Stufe 2 aufgestellt. Aus diesen Anforderungen lassen sich folgende gewünschte Systemeigenschaften ableiten, die von einem Infrastrukturdienst erfüllt werden müssen:

Globale Skalierbarkeit: Die aufzubauende Infrastruktur skaliert im globalen Internet. Dies betrifft die Mengenskalierung der potentiell teilnehmenden Agentenplattformen sowie die Menge der potentiell angebotenen Dienste, die mobile Agenten der Stufe 2 für die Lösung ihres User Tasks in Anspruch nehmen wollen.

Dienstangebot: Das wahrnehmbare Dienstangebot für einen mobilen Agenten der Stufe 2 muss eine ausreichende Menge von Diensten zur Verfügung stellen, damit er seine Aufgabe in autonomer Weise erfüllen kann.

Dienstaktualität: Das für den mobilen Agenten wahrnehmbare Dienstangebot muss so aktuell sein, dass eine Routenplanung und Routenoptimierung sinnvoll möglich ist.

Globale Netzwerkdynamik: Die Dynamik im Internet, die durch Eintreten und Austreten von Endgeräten erzeugt wird und sich durch das Auftauchen und Verschwinden von angebotenen Diensten und Agentenplattformen äußert, muss adäquat beherrscht werden.

Hochdynamische Agentenplattformen: Die Einbindung von hochdynamischen Agentenplattformen, die eventuell nur wenige Sekunden im Agentensystem verweilen, soll unterstützt werden, um mobilen Agenten zur Lösung ihrer Aufgabe die Migration ins Agentennetzwerk zu ermöglichen.

Im nächsten Abschnitt folgt eine Analyse dieser gewünschten Systemeigenschaften, um deren Abhängigkeiten zueinander zu verdeutlichen und die Gesamtmenge der Systemeigenschaften in zusammenhängende Problemfelder aufzuteilen, welche für sich gelöst werden können.

5.2 Analyse und Strukturierung

Im vorherigen Abschnitt wurden die gewünschten Systemeigenschaften an eine Infrastruktur für ein MAS der Stufe 2 aufgestellt, welche alle miteinander im Zusammenhang stehen. Schon im Abschnitt 4.3.4, in welchem die Anforderungen an eine solche Infrastruktur aufgestellt wurden, kamen einige scheinbar im Widerspruch zueinander stehende Forderungen zutage, die im Gesamten schlecht lösbar erscheinen. Aus diesem Grund wird die Menge der Systemeigenschaften in diesem Abschnitt in voneinander unabhängige Teilmengen zerlegt, um eine getrennte Lösung dieser Teilmengen zu ermöglichen, ohne den Gesamtzusammenhang aus dem Fokus zu verlieren.

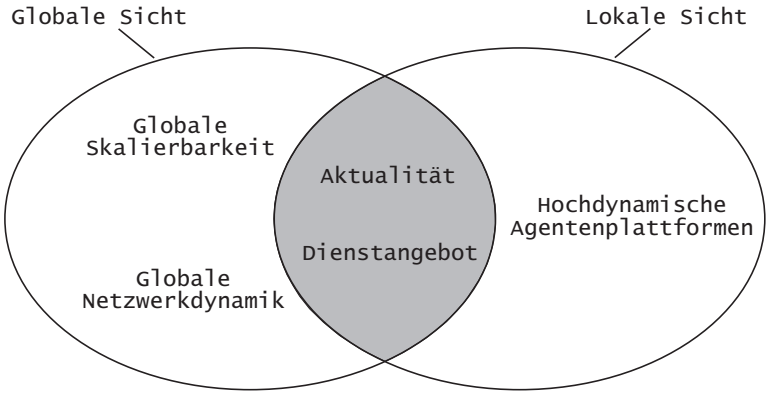

Abbildung 5.1: Isolierte Mengen von Eigenschaften

In Abbildung 5.1 sind die einzelnen Systemeigenschaften in den Teilmengen zusammengefasst, die einzeln betrachtet ein großes Lösungspotential besitzen.

5.2.1 Teilmenge 1 - globale Sicht: Globale Skalierbarkeit - globale Netzwerkdynamik - Dienstangebot - Aktualität

In einem globalen System sollten alle global verfügbaren Dienste verzeichnet sein. Dies betrifft zumindest die Dienste, die ein MA zur Lösung seines User Tasks und damit zur Routenplanung benötigt. Ein MA der Stufe 2 muss daher Dienste unterschiedlich betrachten, je nach dem, in welchem Zustand er sich befindet. Entsprechend der Sichtweise eines MA der Stufe 2 wird daher an dieser Stelle begonnen, Dienste in drei Kategorien zu unterteilen:

1. *Grundsätzliche Dienste eines MAS*,
2. *Infrastrukturdienste* und
3. *Anwendungsdienste*.

Grundsätzliche Dienste eines MAS, wie z.B. das Anbieten von Ressourcen zur Ausführung eines MA, Dienste zur Kommunikation von Agenten untereinander oder zur Migration, die einzig zum technischen Funktionieren eines MAS benötigt werden, können als auf jeder Agentenplattform vorhanden vorausgesetzt werden. Dienste dieser Art befinden sich daher, bezogen auf die Erfüllung des User Tasks eines MA, auf einer niedrigeren Abstraktionsebene, werden nicht zwingend zur autonomen Routenplanung benötigt und brauchen daher auch nicht zwingend in einem global gültigen Dienstverzeichnis verwaltet werden.

Alle Dienste zur Vernetzung von Agentenplattformen und Verwaltung von Dienstinformationen werden *Infrastrukturdienste* genannt. Diese sind zwar nicht direkt zur Erfüllung des User Tasks eines MA notwendig, wohl aber zur Orientierung des MA, da sie die Informationensgrundlage für das autonome Routing schaffen.

Dienste, welche zur Erfüllung des User Tasks benötigt werden, sind anwendungsbezogen und werden deshalb in dieser Arbeit und im Bezug auf die Infrastruktur eines MAS als *Anwendungsdienste*[1] bezeichnet.

Im heutigen Internet werden Anwendungsdienste i.d.R. von Servern angeboten, die einen zuverlässigen und leistungsstarken Computer darstellen. Dies macht auch Sinn, denn ein Anwendungsdienst (z.B. im E-Business Bereich) muss ein relativ zuverlässig anzusprechendes Ziel im Internet darstellen, um akzeptiert

[1] Der hier verwendete Begriff *Anwendungsdienst* deckt sich nicht mit dem Begriff eines *Dienstes der Anwendungsschicht* des OSI- bzw. Internetmodells, der im Kapitel 2.1 eingeführt wurde. Die Dienste der OSI- bzw. Internetmodell-Anwendungsschicht werden den Anwendungsprogrammen vom Betriebssystem zur Verfügung gestellt, z.B. TCP-Sockets oder SSH-Sockets.

und benutzt zu werden. Solche Anwendungsdienste sind nur sehr selten Offline. Dies wird sich auch nicht ändern, wenn die Server zusätzlich in MAS eingebunden werden, um ihre Dienste MA zur Verfügung zu stellen. Natürlich sind auch andere Anwendungsszenarien denkbar, in denen eine gewisse Netzwerkdynamik auftreten kann: So kann ein verteiltes Informationssystem auch private Computer integrieren, die nicht immer, sondern nur unregelmäßig für ein paar Stunden online sind. Ferner können auch mobile Endgeräte eingebunden werden, wie etwa Notebooks oder Smartphones. Es ist aber fraglich, ob von diesen mobilen Endgeräten (z.B. wegen ihrer begrenzten Ressourcen) wirklich Anwendungsdienste zur Nutzung für andere Teilnehmer angeboten werden. Wenn doch, so ist es in jedem Falle wünschenswert, diese Dienste auch zu veröffentlichen. Doch die Dienstinformation über ein Dienstangebot, welches nur ein paar Minuten zur Verfügung steht, weil das mobile Endgerät samt Agentenplattform dann schon wieder aus dem MAS verschwunden ist, veraltet vielleicht schon, bevor sie im gesamten globalen System verteilt ist. Daher sollten im globalen System nur solche Anwendungsdienstinformationen verteilt werden, die eine gewisse Stabilität und Langlebigkeit erwarten lassen und damit die Gefahr einer fehlerhaften Routenplanung auf Grund nicht aktueller Dienstinformationen verringert. Ein Infrastrukturdienst für MAS der Stufe 2 sollte, bezogen auf die globale Sicht auf das Gesamtsystem, alle diese Eigenschaften erfüllen.

5.2.2 Teilmenge 2 - lokale Sicht: Hochdynamische Agentenplattformen - Dienstangebot - Aktualität

Hochdynamische Agentenplattformen, die z.B. auf mobilen Endgeräten gehostet werden, können unter Umständen nur wenige Sekunden im MAS verweilen. Diese Zeit würde ausreichen, einen schon programmierten und auf die Möglichkeit zur Migration wartenden MA in das MAS migrieren zu lassen. Da der MA der Stufe 2 autonom handelt, muss er in dieser Situation entscheiden, was er tut. Er hat zwei Alternativen: Entweder er migriert sofort auf eine andere Agentenplattform, weil diese ihm eine stabilere Ausführungsumgebung bietet als seine momentane, oder er wartet auf die Übertragung von aktuellen Anwendungsdienstinformationen, um seine Routenplanung auf seiner aktuellen hochdynamischen Plattform auszuführen und dann erst zu migrieren. Die Anwendungsdienstinformationen können eventuell aus einer relativ großen Datenmenge bestehen und die Übertragung an die hochdynamische Agentenplattform könnte mehr Zeit in Anspruch nehmen, als sie insgesamt im Netzwerk verbleibt. In diesem Fall wäre eine Entscheidung des MA zu einer sofortigen Migration auf eine andere potentere Agentenplattform die bessere Wahl. Der MA braucht für seine Entscheidung aber nur Infrastrukturinformationen, also Informationen, welche Agentenplattformen in seiner unmittelbarer Nähe existieren und ob diese geeignet sind, sofort

dorthin zu migrieren. Die lediglich aus der lokalen Netzwerkumgebung[2] des MAS stammenden Infrastrukturinformationen, die nur aus einer sehr begrenzten Datenmenge bestehen, können getrennt von den Anwendungsdienstinformationen übertragen werden und somit viel schneller zur Verfügung stehen.

Um einen MA in dieser Situation angemessen zu unterstützen, muss ein Infrastrukturdienst deshalb dafür Sorge tragen, dass an eine neu in das MAS eingetretene Agentenplattform zuerst und sehr schnell Infrastrukturinformationen übertragen werden. Weiterhin braucht der MA Informationen über das Potential (z.B. Langlebigkeit, Rechenleistung, Speicherplatz) seiner aktuellen und der anderen, ihn unmittelbar umgebenden Agentenplattformen, um bei einem autonomen Entscheidungsprozess eine geeignete Plattform auswählen zu können. Ferner muss eine potentiell geeignete Plattform auch die entscheidenden Routing-Dienste anbieten, damit sich der MA nach der Migration auf diese sofort seine Route planen lassen kann. Mit den Infrastrukturinformationen müssen somit auch Informationen über Routing-Dienste, die MAS zu einem MAS der Stufe 2 qualifizieren, mit übertragen werden.

Verbleibt eine hochdynamische Agentenplattform länger als ein paar Sekunden im MAS, lohnt es sich, ihre angebotenen Anwendungsdienste zu veröffentlichen. Dienstinformationen über Dienstangebote, welche nur wenige Minuten zur Verfügung stehen, veralten vielleicht schon, bevor sie im globalen System verteilt sind. Es lohnt sich daher nicht, sie global zu veröffentlichen, wohl aber im örtlich unmittelbaren, lokalen Netzwerkbereich eines MAS. Von diesen Anwendungsdiensten können so wenigstens die sich in relativer Nähe zum Dienst befindlichen MA profitieren. Ein Infrastrukturdienst für MAS der Stufe 2 sollte, bezogen auf die lokale Sicht auf das Gesamtsystem, alle diese Eigenschaften erfüllen.

5.2.3 Schnittmenge der Teilmengen 1 und 2 - Manager: Dienstangebot - Aktualität

Die zuvor beschriebenen Teilmengen betonen jeweils eine andere Sichtweise auf ein MAS-Gesamtsystem: Teilmenge 1 betont die Sicht auf das globale MAS, Teilmenge 2 betont die Sichtweise auf die lokale Netzwerkumgebung eines MAS. In Teilmenge 1 wurde die Unterscheidung von Diensten nach unterschiedlichen Abstraktionsebenen in Anwendungsdienste, Infrastrukturdienste und grundsätzliche Dienste eines MAS beschrieben. In Teilmenge 2 wurden die Anwendungsdienste noch einmal unterschiedlich behandelt: Langfristig verfügbare Anwendungsdienste sollen global veröffentlicht werden, nur kurzfristig verfügbare Anwendungsdienste sollen nur im lokalen Netzwerkbereich eines MAS veröffentlicht

[2]Hier ist die lokale Netzwerkumgebung auf Ebene des MAS gemeint und nicht die Umgebung des lokalen Netzwerkes (LAN), wie sie im Abschnitt 2.2.1 beschrieben wird.

werden. Informationen über Infrastrukturdienste und über grundsätzliche Dienste eines MAS spielen nur im lokalen Netzwerkbereich eines MAS eine Rolle und können dort deutlich performanter gemanagt werden. An der Schnittstelle dieser beiden Sichtweisen entsteht der Bedarf an einer Komponente, welche die beiden Sichten intelligent miteinander verbindet. Die Aufgabe dieser Managerkomponente muss darin bestehen, die Informationen über Anwendungsdienste im lokalen Netzwerkbereich eines MAS zu verwalten und z.B. die Entscheidung zu treffen, ob und wann ein Anwendungsdienst global veröffentlicht wird. Die Plattform, auf der der Manager wirkt, muss stabile Bedingungen bieten. Sie stellt einen relativ fixen, zentralen Punkt dar - ein Widerspruch zur Netzwerkdynamik. Daher muss sichergestellt werden, dass es möglich ist, die Managerrolle dynamisch auch von anderen Plattformen im lokalen Netzwerkbereich annehmen zu lassen. Dazu sind Informationen über geeignete potentielle Plattformen nötig, also die gleichen Informationen, wie sie in Teilmenge 2 im Zusammenhang mit der Migrationsentscheidung eines MA schon beschrieben wurden. Der lokale Netzwerkbereich einer Infrastruktur eines MAS muss also auf Dynamik reagieren können, der Manager muss in autonomer Weise dynamisch wählbar und zur Laufzeit austauschbar sein. Ein Infrastrukturdienst für MAS der Stufe 2 sollte alle diese Eigenschaften erfüllen, um die verschiedenen Sichtweisen auf das Gesamtsystem zu verbinden.

5.3 Umsetzungsvorschlag

In den vorherigen Abschnitten wurden aus den Anforderungen an die Infrastruktur die gewünschten Systemeigenschaften abgeleitet und in zusammenhängende Teilmengen strukturiert. Aus den Teilmengen von Systemeigenschaften wurden drei Aufgabenbereiche abgeleitet, die einzeln betrachtet ein großes Lösungspotential versprechen. Aus den drei Aufgabenbereichen lassen sich drei Infrastrukturdienstkomponenten ableiten, deren funktionale Gesamtheit ein Rahmenwerk bildet, welches alle Anforderungen und Aufgaben einer Infrastruktur für ein MAS der Stufe 2 erfüllt. Teilmenge 1 führt zu einer Komponente, welche die globale Sicht auf das Gesamtsystem abdeckt. Teilmenge 2 führt zu einer Komponente, welche die Sicht auf den hochdynamischen lokalen Netzwerkbereich abdeckt. Die Schnittmenge der beiden Teilmengen führt zur Managerkomponente, welche die beiden anderen Infrastrukturkomponenten intelligent miteinander verbindet. Abbildung 5.2 zeigt den Zusammenhang der einzelnen Infrastrukturkomponenten als Rahmenwerk und wie sie sich in die gesamte Architektur zwischen MAS und Routingdiensten integrieren.

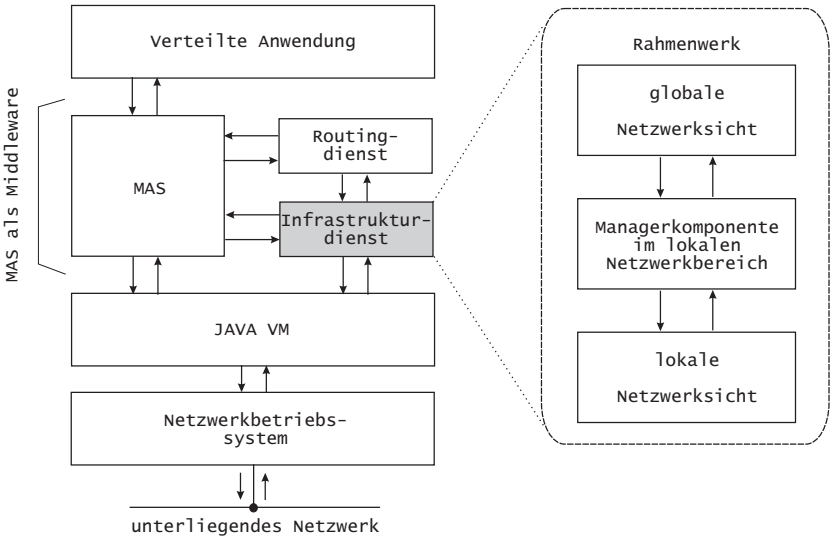

Abbildung 5.2: Mögliche Architektur eines Gesamtsystems und Integration der Infrastrukturdienste

5.4 Thesen

These 1 - Struktur: Die Trennung des gesamten mobilen Agentensystems in eine globale und eine lokale Sicht hilft, alle Anforderungen an eine Infrastruktur für MAS der Stufe 2 zu erfüllen. Für die Verbindung der beiden Sichten ist eine intelligente Komponente nötig, welche die autonome Verwaltung des lokalen Bereiches sicherstellt und eine Selbstadaption des Netzwerkes an dynamische Vorgänge ermöglicht.

These 2 - Dynamik: Mit einer lokalen Sicht auf ein mobiles Agentensystem der Stufe 2 ist es möglich, auch hohe Netzwerkdynamik performant abzubilden. Darüber hinaus brauchen mobile Agenten der Stufe 2 eine intelligente Unterstützung seitens ihrer umgebenden Infrastruktur, die über die gewöhnliche Transparenz im Peer-to-Peer-System hinaus geht.

These 3 - Skalierbarkeit: Die globale Sicht auf ein mobiles Agentensystem hilft, die globale Skalierbarkeit des Gesamtsystems zu beherrschen. Diese kann durch einen DHT-basierten Ansatz realisiert werden.

These 4 - Praktikabilität: Die Implementierung der genannten Komponenten (siehe Abbildung 5.2) ist möglich, ohne Schnittstellen zu vorhandenen Systemen durchbrechen zu müssen. Darüber hinaus ist es für eine der Komponenten sogar möglich, schon vorhandene Technologien zu nutzen.

Im nächsten Kapitel wird der in Abschnitt 5.3 vorgestellte architekturelle Umsetzungsvorschlag eines Infrastrukturdienst-Frameworks für ein MAS der Stufe 2 wieder aufgegriffen und dessen Umsetzung in konkrete Infrastrukturdienstkomponenten beschrieben. Anschließend werden in Teil III dieser Arbeit die Komponenten evaluiert.

6 Übersicht und allgemeine Beschreibung von QuickLinkNet

In diesem Teil der Arbeit wird das Infrastrukturdienst-Framework *QuickLinkNet* vorgestellt und beschrieben. Das Framework folgt der in Kapitel 5 vorgestellten Architektur und stellt eine konkrete Umsetzung des Lösungsvorschlags aus demselben Kapitel dar. QuickLinkNet besteht aus den drei Infrastrukturdiensten

- *QuickLink* (lokale Sicht),
- *ServiceJuggler* (Manager) und
- *APLICOOVER* (globale Sicht),

die gemeinsam die Anforderungen an einen komplexen Infrastrukturdienst für ein mobiles Agentensystem der Stufe 2 erfüllen. Dieses Kapitel beschreibt die Gesamtarchitektur des Infrastrukturdienst-Frameworks und dessen Aufgabe gegenüber einem MAS (oder anderen verteilten Anwendungen) und den Routingdiensten. Die wichtigsten Begriffe im Zusammenhang mit QuickLinkNet werden vorgestellt. Ferner wird der Aufbau des Overlaynetzwerkes, das aus dem Zusammenspiel der Infrastrukturkomponenten QuickLinkNets entsteht, dargestellt.

Die einzelnen Infrastrukturdienste als Komponenten des Frameworks werden in den folgenden Kapiteln 7 bis 9 detailliert beschrieben.

6.1 Architektur und Zusammenspiel der Komponenten

QuickLinkNet bildet die organisatorische Hülle um die drei vorgestellten Infrastrukturdienste, welche ihrerseits die Grundkomponenten für QuickLinkNet darstellen. Betrachtet man QuickLinkNet von außen aus Sicht eines MAS oder der Routingdienste, so bildet es in seiner funktionalen Gesamtheit einen komplexen Infrastrukturdienst, welcher die Anforderungen eines MAS der Stufe 2 erfüllt.

Die einzelnen Komponenten sind für sich gesehen aber einfachere Infrastrukturdienste, die auch für sich funktionstüchtig und ansprechbar sind. Dazu wurden sie modular programmiert, um nach Bedarf gestartet werden zu können. Zur Kommunikation bietet jede Komponente ihre Schnittstellen unabhängig von den anderen Komponenten an. Die Komponenten sind, wie die meisten MAS auch,

in der Programmiersprache Java 5.0 [Sun07b] implementiert, um eine möglichst hohe Praktikabilität bei ihrer Einbindung in MAS und andere komplexe Dienste sicherzustellen. Einen Überblick über die Komponenten QuickLinkNets und deren Einbindung in die Gesamtarchitektur mit einem MAS und den Routingdiensten zeigt Abbildung 6.1. Dabei kommen den einzelnen Diensten folgende Aufgaben zu:

1. Die lokale Netzwerksicht[1] übernimmt in *QuickLinkNet* die konkrete Komponente *QuickLink*. QuickLink verwaltet einen *lokalen Teilbereich eines logischen Netzwerkes* und ist u.a. für das schnelle Einbinden von mobilen Agentenplattformen zuständig. Ebenfalls sorgt es für einen zeitnahen Überblick über alle im lokalen Netzwerkbereich angebotenen *Infrastrukturdienste* eines MAS. Zusätzlich erfüllt QuickLink alle Aufgaben, die im Zusammenhang mit der *leistungsbezogenen Charakterisierung einer Agentenplattform* stehen. QuickLink stellt den grundlegendsten Infrastrukturdienst von QuickLinkNet dar und muss auf jeder Agentenplattform ausgeführt werden, welche in das logische Netzwerk und damit in ein MAS eingebunden werden soll.

2. Die zweite Softwarekomponente, welche die Aufgabe des *Managers im lokalen Teilbereich eines logischen Netzwerkes*[2] übernimmt, ist *ServiceJuggler*. Diese Komponente sorgt für die *Verwaltung von Anwendungsdienstinformationen im lokalen Netzwerkbereich* und übernimmt die *Kommunikation mit der Komponente für die globale Netzwerksicht (APLICOOVER)*. Ferner steuert sie die *Adaption der Agentenplattformen im lokalen Netzwerkbereich* eines MAS als Reaktion auf dynamische Vorgänge. Zur Steuerung der Adaption bedient sich ServiceJuggler wiederum leistungsbezogener Informationen, die von *QuickLink* geliefert werden. Die Adaption selbst erfolgt durch die dynamische Migration von bestimmten Infrastrukturdiensten auf geeignete Agentenplattformen, die dadurch spezielle Rollen im gesamten MAS annehmen.

3. *APLICOOVER* stellt die dritte Infrastrukturdienstkomponente in *QuickLinkNet* dar und realisiert die *globale, anwendungsdienstbezogene Sicht*[3] auf das MAS. Es dient dazu, die *logischen lokalen Netzwerke*, die durch QuickLink gebildet und durch ServiceJuggler verwaltet werden, *zu verbinden* und die *globale Skalierbarkeit des Gesamtsystems* sicherzustellen. Mobile Agenten und Agentenplattformen erhalten durch APLICOOVER die Möglichkeit, jeden im globalen MAS propagierten Anwendungsdienst zu finden.

[1] Vgl. dazu auch Abschnitt 5.2.2 und Abbildung 5.2.
[2] Vgl. dazu auch Abschnitt 5.2.3 und Abbildung 5.2.
[3] Vgl. dazu auch Abschnitt 5.2.1 und Abbildung 5.2.

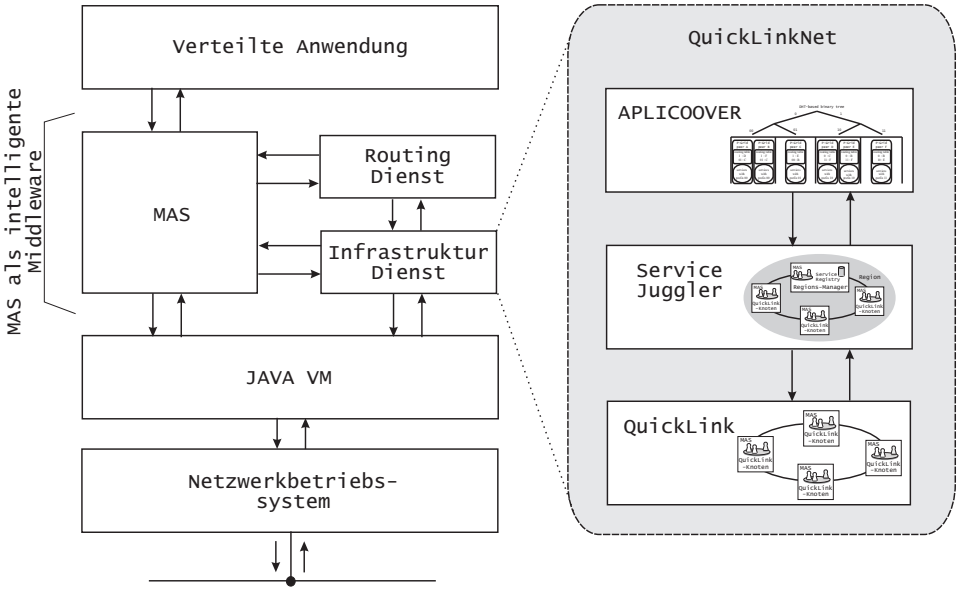

Abbildung 6.1: Die Architektur von QuickLinkNet im Zusammenspiel mit MAS und Routingdiensten

Damit die funktionale Unabhängigkeit aller drei Infrastrukturdienste QuickLinkNets sichergestellt ist, arbeiten sie unabhängig voneinander direkt auf den Kommunikationsdiensten der Java VM. Im Zusammenspiel von MAS, Infrastrukturdiensten und Routingdiensten entsteht ein MAS der Stufe 2, welches, wie in Abbildung 6.1 dargestellt, gegenüber einer verteilten Anwendung die Rolle einer intelligenten Middleware einnimmt.

6.2 Spezielles Vokabular

Aus dem Zusammenwirken der Infrastrukturdienste QuickLinkNets entsteht ein logisches, strukturiertes Overlaynetzwerk, dessen Aufbau im nächsten Abschnitt vorgestellt wird. Das logische Netzwerk selbst besteht aus Infrastrukturelementen, die inhaltlich schon angedeutet wurden und jetzt näher definiert werden:

QuickLink-Knoten: Eine Agentenplattform, die den Infrastrukturdienst QuickLink ausführt, wird im Zusammenhang mit der logischen Vernetzung *QuickLink-Knoten (QuickLink node)* genannt. Da QuickLink den grundlegendsten Infrastrukturdienst innerhalb QuickLinkNets erfüllt, der zwingend auf allen in das Netzwerk eingebundenen Agentenplattformen ausgeführt werden muss, ist jede über QuickLink eingebundene Agentenplattform automatisch ein QuickLink-Knoten. Er stellt die kleinste Struktureinheit im

logischen Netzwerk dar.

Region: Eine *Region (region)* bezeichnet eine logisch zusammenhängende Gruppe von QuickLink-Knoten, die durch den Infrastrukturdienst QuickLink gebildet wird. Der Begriff entspricht dem im MASIF-Standard [The00], in welchem er eine Menge von Agentenplattformen bezeichnet, die der gleichen Autorität unterstehen[4], also einen administrativ abgeschlossenen Bereich bilden. Innerhalb QuickLinkNets beschränkt sich eine Region auf den Bereich des unterliegenden IP-Subnetzes, der i.d.R. einer einzigen Autorität untersteht.

Regions-Manager: *(region manager)* Dieser Begriff bezeichnet einen QuickLink-Knoten innerhalb einer Region, der zusätzliche Aufgaben gegenüber anderen QuickLink-Knoten erfüllt und dadurch eine besondere Rolle wahrnimmt. Er verwaltet die Informationen aller Anwendungsdienste, die in der Region bekannt sind. Dazu betreibt der Regions-Manager einen Verzeichnisdienst für Anwendungsdienste (der *ServiceRegistry*), an dem alle in der Region propagierten Anwendungsdienste registriert werden. Außerdem übernimmt er für die ganze Region die Verbindung zum globalen Netzwerk und die Suche nach Anwendungsdiensten in diesem. Der Regions-Manager kann seine Rolle auf einen anderen QuickLink-Knoten der Region übertragen, wenn dies dynamische Veränderungen in der Region erfordern. Dazu migriert er der ServiceRegistry mit all seinen Informationen. Prinzipiell stellt der Regions-Manager Mechanismen zur Verfügung, um auch andere Anwendungsdienste, welche diese Möglichkeit unterstützen, migrieren zu lassen.

ServiceRegistry: Der ServiceRegistry ist hauptsächlich ein Verzeichnisdienst für Anwendungsdienste (ein *service directory*) und beinhaltet alle verfügbaren Informationen über Anwendungsdienste, die innerhalb der Region bekannt sind und potentielle Zielpunkte für mobile Agenten darstellen. Eine Agentenplattform, die Anwendungsdienste anbieten will, muss diese am ServiceRegistry anmelden.

Regionsprofil: Als Regionsprofil werden die vom Regions-Manager aus den Informationen des ServiceRegistry erzeugten, kummulierten Anwendungsdienstinformationen bezeichnet, die dieser an das globale System weiterreicht und dort veröffentlicht.

Globales System: Das globale System verbindet alle Regionen auf der Basis ihrer propagierten Anwendungsdienste miteinander. Es besteht aus Regions-Managern, die ihre im ServiceRegistry registrierten Dienste in Verbindung

[4] Vgl. dazu Abschnitt 4.3.5.1.

mit ihrer Lokationsinformation (Regionsprofil) im globalen System veröffentlichen. Zur Verbindung zum globalen System muss auf einem Regions-Manager eine Instanz von APLICOOVER gestartet sein, welche die Einbindung in das globale System übernimmt.

Neben den Infrastrukturelementen, aus denen das logische Netzwerk gebildet wird, bedarf es zur Erklärung der Funktionsweise der Overlaystruktur noch folgender Begriffe:

Priorität: *(priority)* Dieser Begriff wurde schon im Domain-Management von TRACY [BER01] verwendet und stellt dort einen Zahlenwert dar, der die leistungsbezogene Eignung einer Plattform für besondere Zwecke, relativ zu anderen Agentenplattformen gesehen, festlegt. In QuickLinkNet existieren mehrere Prioritätswerte, von denen einer oder mehrere für die Prüfung der Eignung verwendet werden können. Ein sinnvoller Prioritätswert zur Steuerung einer Region ist die Anwesenheitszeit einer Agentenplattform im logischen Netzwerk; es können aber auch der zur Verfügung stehende Speicherplatz, die Rechenleistung und andere Werte herangezogen werden. Die Bildung der Prioritätswerte und deren Verwendung wird im Kapitel 7 detailliert beschrieben.

PerformanceAnalyser: Ist eine in QuickLink integrierte Softwarekomponente, die verschiedene Leistungsparameter einer Agentenplattform misst. Die Performancewerte werden über Schnittstellen zur Verfügung gestellt und gleichzeitig in aufbereiteter Form als Prioritätswerte wiedergegeben. Der PerformanceAnalyser wird ebenfalls in Kapitel 7 detailliert beschrieben.

Zyklusnachricht: *(cycle message)* Eine Broadcastnachricht, die zur Verwaltung einer Region von QuickLink-Knoten verwendet wird. Die Zyklusnachricht enthält Informationen über den sendenden QuickLink-Knoten selbst und über alle ihm bekannten QuickLink-Knoten seiner Region. Sie beinhaltet die Leistungsparameter des sendenden QuickLink-Knotens und die Adressen sowie Informationen über die Infrastrukturdienste und grundlegenden Dienste des MAS aller bekannten QuickLink-Knoten der Region.

Beitrittsnachricht: *(join message)* Eine Broadcastnachricht, die zum Zwecke des Beitritts eines QuickLink-Knotens zu einer Region von ihm ausgesendet wird und Informationen über seine Infrastrukturdienste, die grundlegenden Dienste seines MAS, seine Leistungsparameter und seine Adresse enthalten.

Aktualisierungsnachricht: *(update message)* Eine Broadcastnachricht, die ein Quick-Link-Knoten zur sofortigen Propagation der Änderung eines seiner Infrastrukturdienste oder Leistungsparameter aussenden kann.

Der genaue Aufbau der Nachrichten wird in Kapitel 7 beschrieben. Nachdem nun die Aufgaben der Komponenten QuickLinkNets erklärt und die verwendeten Infrastrukturelemente eingeführt wurden, wird im nächsten Abschnitt die von QuickLinkNet aufgebaute Overlaystruktur mit ihren leistungsbezogenen Eigenschaften dargestellt.

6.3 Das logische Netzwerk

6.3.1 Aufbau des Netzwerkes

QuickLinkNet formt ein logisches Netzwerk, welches in zwei Ebenen geteilt ist. Die *untere Ebene* besteht aus Regionen, welche ihrerseits aus QuickLink-Knoten bestehen. In jeder Region existiert ein QuickLink-Knoten, der als Regions-Manager fungiert. Die *obere Ebene* besteht aus Regions-Managern, die als Stellvertreter für je eine ganze Region auftreten. Der Aufbau des so gebildeten logischen Netzwerkes ist in Abbildung 6.2 dargestellt.

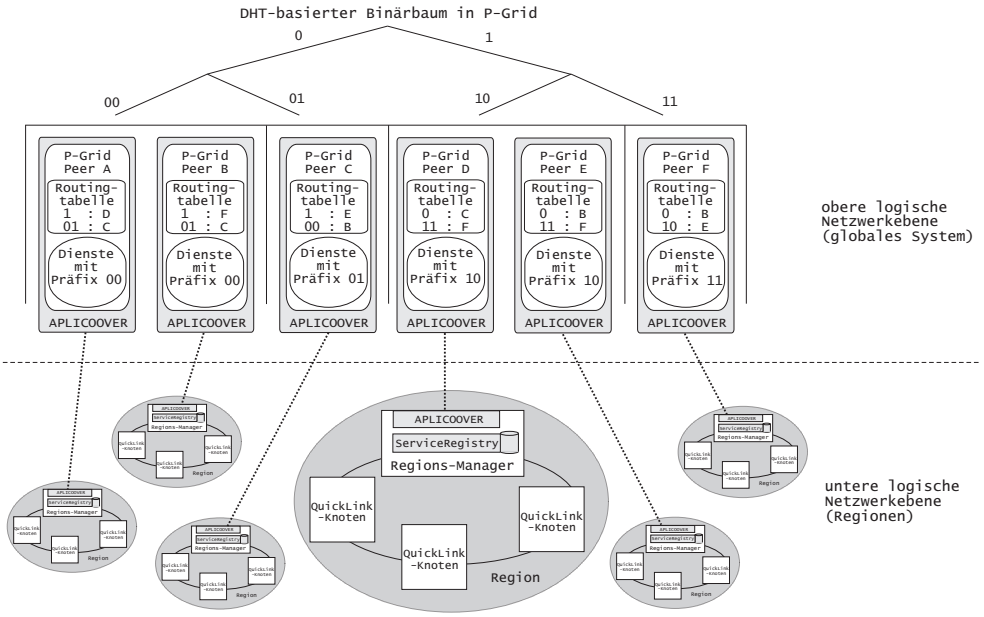

Abbildung 6.2: Das von QuickLinkNet aufgespannte logische Netzwerk teilt sich in zwei Ebenen

Die beiden Ebenen des logischen Netzwerkes unterscheiden sich in der Abstraktion von den unterliegenden Agentenplattformen, in der Form der verwendeten Overlaystrukturen, und, daraus folgend, der Aktualität, der abbildbaren Dynamik und der Skalierbarkeit der verwendeten Informationen. Die Eigenschaften

werden in den folgenden Unterabschnitten erläutert. Die Regions-Manager sind in beiden Ebenen des Netzwerkes präsent. Dabei nehmen sie, an der Nahtstelle der beiden Ebenen sitzend, eine vermittelnde Rolle ein.

6.3.1.1 Die untere Netzwerkebene

Abstraktionsgrad: Auf der unteren Ebene des logischen Netzwerkes wird von der Topologie des unterliegenden technischen Netzwerkes abstrahiert. Bezüglich der Agentenplattform als technischem Infrastrukturelement herrscht aber eine geringe Abstraktion. Die QuickLink-Knoten werden, wie in Abbildung 6.3 dargestellt, als Agentenplattformen inklusive ihrer Leistungseigenschaften betrachtet. Diese zusätzlichen Informationen gehen über das gewöhnliche Maß der Transparenz von Peers in Peer-to-Peer-Systemen hinaus, bei denen sonst gerade die Gleichwertigkeit der Peers betont wird.

QuickLink-Knoten besitzen Informationen über ihre technische Leistungsfähigkeit, die sie mit Hilfe ihres PerformanceAnalysers ermitteln. Sie kennen auch ihre aktuell ausgeführten Infrastrukturdienste und die grundlegenden Dienste des MAS[5], welches sich auf dem QuickLink-Knoten befindet. Alle diese Informationen benötigt

1. ein MA, der schnell auf eine andere Agentenplattform mit hoher Stabilität migrieren möchte, um bessere Voraussetzungen für eine Routenplanung zu bekommen, und
2. der Regions-Manager der Region, um die Verwaltung der Anwendungsdienstinformationen an die jeweilige Netzwerksituation zu adaptieren.

Die an dieser Stelle (noch) nicht nötigen Informationen über die Anwendungsdienste selbst werden auf dem Regions-Manager gesammelt und verwaltet. Dazu wird ein anderer Mechanismus verwendet als bei der Verwaltung der Informationen über Infrastruktur- und grundlegende Dienste der Agentenplattformen.

Overlaystruktur: Das Overlaynetzwerk auf der unteren Netzwerkebene wird durch den Infrastrukturdienst QuickLink gebildet. Aufgrund der leistungsmäßigen Anforderungen hochdynamischer Agentenplattformen[6] wurde ein *logischer Ring* als Overlaystruktur gewählt. Die teilnehmenden QuickLink-Knoten senden nacheinander in vorgegebener Reihenfolge *Zyklusnachrichten* in regelmäßigen Abständen t_{cycle} in das lokale Netzwerk, welche von

[5] Diese Informationen werden nicht vom Funktionsumfang des Infrastrukturdienst-Frameworks QuickLinkNet erzeugt, sondern müssen vom MAS über Schnittstellen bereit gestellt werden.

[6] Vgl. hierzu die Anforderungen an diese Komponente in Abschnitt 5.2.2 und die Aufgaben der konkreten Komponente QuickLinkNet in Abschnitt 6.1.

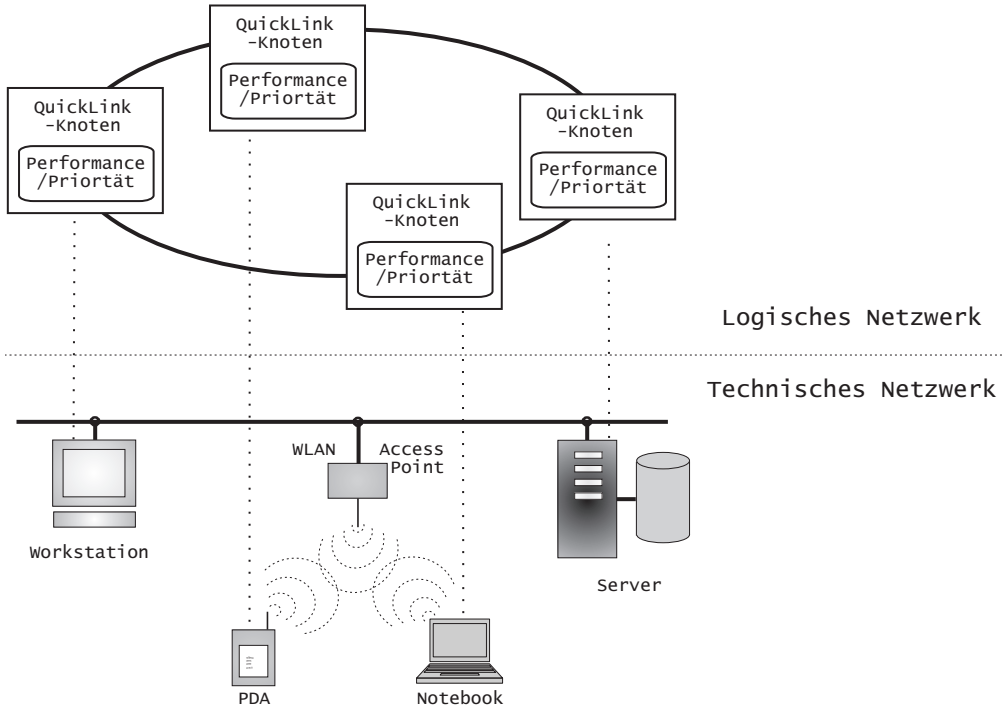

Abbildung 6.3: Untere Netzwerkebene: QuickLink abstrahiert von der netzwerktechnischen Topologie im lokalen IP-Netzwerkbereich, nicht aber von der Leistungsfähigkeit der Endgeräte

allen QuickLink-Knoten empfangen und ausgewertet werden. Hat der letzte QuickLink-Knoten gesendet, fängt der erste wieder von vorn an. Die genaue Funktionsweise wird in Kapitel 7 beschrieben.

Dynamik und Aktualität der Informationen: Die Dynamik der in der Region verwalteten Agentenplattformen kann sehr hoch sein und im Sekundenbereich liegen. Die Verwaltung der Infrastrukturdienstinformationen muss dem daraus folgenden Anspruch an die Aktualität entsprechen können. QuickLink verwendet daher auch Broadcastnachrichten *(Beitrittsnachrichten)* zur Einbindung von Agentenplattformen in eine Region. Beitrittsnachrichten sendet ein isolierter QuickLink-Knoten regelmäßig alle t_{join} über sein Netzwerkinterface. Wird eine solche Nachricht bei seinem Netzeintritt von allen in diesem Netzwerk vorhandenen Quick-Link-Knoten empfangen, so sind diese alle gleichzeitig von der Anwesenheit des neuen Knotens informiert und können gleichzeitig jeweils ihre eigene Informationsbasis aktualisieren. Mit der nächsten Zyklusnachricht werden alle QuickLink-Knoten mit aktualisierten Informationen versorgt und können diese mit ihrer eige-

nen Informationsbasis vergleichen. Auch der eintrittswillige Knoten erhält diese Zyklusnachricht. Mit der aktuellen Informationsbasis der Region versorgt, kann er nun überprüfen, ob er selbst in die Region eingebunden wurde. Im positiven Fall sendet er nun keine Beitrittsnachrichten mehr, im negativen Fall sendet er erneut eine Beitrittsnachricht. Die Einbindung t_{insert} eines neuen QuickLink-Knotens kann nach diesem Ablauf in der Zeit $t_{insert} = t_{join} + t_{cycle}$ geleistet werden. Bei entsprechend kurz gewählten Zeitintervallen liegt die Eintrittszeit im Bereich weniger Sekunden.

Die Verwaltung der Anwendungsdienstinformationen unterliegt nicht den gleichen zeitlichen Skalierungsanforderungen wie die der Infrastrukturdienste, denn die Propagation von Anwendungsdiensten ist nur dann sinnvoll, wenn sie für eine gewisse Zeit stabil ansprechbar und erreichbar sind. Die Verwaltung von Anwendungsdiensten wird deshalb in einer Region zentral vom Regions-Manager geleistet. Bei der angenommenen mäßigeren Dynamik der Anwendungsdienstinformationen in einer Region von mehreren 10 Sekunden ist die zentrale Datenhaltung sinnvoll, um eine regionweit konsistente Sicht auf die Anwendungsdienste zu gewährleisten.

Skalierbarkeit: Auf der unteren Netzwerkebene spielt die Mengenskalierbarkeit bezüglich des potentiellen Dienstangebotes keine so ausgeprägte Rolle wie in der oberen Netzwerkebene, die das globale Netzwerk umfasst. Aufgrund der dynamischen Anforderungen müssen in der unteren Netzwerkebene vielmehr die zeitliche Skalierbarkeit und die Menge der zur Verwaltung benötigten Nachrichten beachtet werden. QuickLink verwendet deshalb für die Verwaltung der Region Broadcastnachrichten (Zyklusnachrichten). Durch diese können mit einer Nachricht alle QuickLink-Knoten erreicht und die Netzwerklast gering gehalten werden. Um die Größe der Nachricht klein zu halten, darf eine solche Verwaltungsnachricht nur geringe Informationsmengen und/oder sehr verdichtete Informationen beinhalten. Der Aufbau und der Inhalt von QuickLink-Nachrichten wird in Kapitel 7 ausführlich behandelt.

6.3.1.2 Die obere Netzwerkebene

Abstraktionsgrad: Auf der oberen Ebene des logischen Netzwerkes herrscht ein höherer Abstraktionsgrad als auf der unteren Netzwerkebene. Hier werden Agentenplattformen nur noch in Bezug auf die Anwendungsdienste, die sie ausführen, betrachtet. Die Anwendungsdienste und die Informationen, wie sie im Netzwerk angeboten werden, dienen als Zielpunkte für mobile Agenten der Stufe 2. Auf dieser Netzwerkebene wird also die User-Task bezogene Sicht eines mobilen Agenten auf das Netzwerk betont, die zur Routenplanung nötig ist. Die tatsächlichen Leistungseigenschaften der Agentenplatt-

formen, auf welcher ein gesuchter Anwendungsdienst ausgeführt wird, kann an dieser Stelle vernachlässigt werden, da er für die globale Routenplanung keine Rolle spielt.

Eine zusätzliche Abstraktion im globalen System erfolgt durch die Stellvertreterfunktion der Regions-Manager, die ihre ganze Region repräsentieren. Dazu verdichtet der Regions-Manager alle Anwendungsdienstinformationen, die er verwaltet, und verknüpft sie mit seiner eigenen Adresse. Für einen mobilen Agenten sind die Anwendungsdienste einer anderen Region im globalen System daher als Anwendungsdienste einer Region und nicht der einer bestimmten Agentenplattform sichtbar. Dies scheint auf den ersten Blick ein Nachteil für einen MA zu sein, weil es eine zusätzliche Migration innerhalb der Zielregion bedeutet, also einen zusätzlichen Aufwand. Unterstellt man aber eine relativ hohe Dynamik innerhalb einer Region, so macht es Sinn, wenn sich ein MA, der gerade frisch in eine Region migriert ist, sich zuerst mit den detaillierten Informationen der Region versorgt und seine Zielpunkte und Routenplanung aktualisiert. Auch Erfurth befürwortet in seiner Dissertation [Erf04] diese Vorgehensweise: Da ein mobiler Agent während der Migration eingefroren ist und daher den zeitlichen Ablauf nicht nachvollziehen kann, können sich nach seinem Erwachen in der neuen Region die Umgebungsbedingungen, auf denen seine Planung basiert, geändert haben. Im Gesamtablauf der Erfüllung des User-Tasks muss daher eine zusätzliche Migration auf der Basis aktueller Daten nicht unbedingt von Nachteil sein, weil sie die Gefahr einer fehlerhaften Route verringern und damit die Arbeitseffizienz eines MA verbessern kann.

Overlaystruktur: Die obere Netzwerkebene strukturiert das globale Netzwerk. Die Anforderungen auf dieser Ebene[7] erfordern vor allem die Mengenskalierung der Anwendungsdienstinformationen. Als geeignete Overlaystruktur wurde ein binärbaumbasierter Ansatz gewählt, der die geforderten Eigenschaften hinreichend gut erfüllt[8]. APLICOOVER kapselt die Funktionalität des gewählten Ansatzes P-Grid [Abe01] und stellt sie in Form von angepassten Diensten für QuickLinkNet zur Verfügung.

Dynamik und Aktualität der Informationen: Neben der Mengenskalierbarkeit ist die abbildbare Dynamik im globalen System zwar wichtig, stellt aber aufgrund der erwünschten relativen Stabilität der Anwendungsdienste einen weniger kritischen Faktor als in der unteren Netzwerkebene dar. Um die Verringerung der Dynamik in der oberen Netzwerkebene zu unterstützen, repräsentiert ein Regions-Manager eine ganze Region und nicht nur

[7] Vgl. hierzu die Anforderungen an diese Komponente in Abschnitt 5.2.1 und die Aufgaben der konkreten Komponente APLICOOVER in Abschnitt 6.1.

[8] Vgl. hierzu die Zusammenfassung Peer-to-Peer-Systeme, Abschnitt 4.2.5.

eine einzelne Plattform. Eine Region im logischen Netzwerk QuickLinkNets als Ganzes ist meist stabiler als eine einzelne Agentenplattform. Sie kann auf Netzwerkdynamik reagieren, um die Robustheit und Zuverlässigkeit einer Region zu erhöhen. Ferner propagiert der Regions-Manager nur solche Anwendungsdienste im globalen System, die eine gewisse Zuverlässigkeit bewiesen haben. Die abbildbare Dynamik im globalen System erreicht mit der gewählten, P-Grid-basierten Lösung den Bereich mehrerer Minuten.

Skalierbarkeit: Die Mengenskalierbarkeit der Anwendungsdienstinformationen auf der oberen Netzwerkebene wird durch APLICOOVER sichergestellt. Es basiert, wie schon unter dem Punkt Overlaystruktur erwähnt, auf einem binärbaumbasierten Ansatz namens P-Grid, welcher die Mengenskalierbarkeit in Verbindung mit der erforderlichen Dynamik leistet. Es könnten auch andere Ansätze verwendet werden, die die gleiche oder eine höhere Leistungsfähigkeit besitzen.

6.3.2 Zusammenwirken der Infrastrukturelemente

QuickLink-Knoten betreiben mindestens den Infrastrukturdienst QuickLink. Die QuickLink-Instanzen auf den QuickLink-Knoten kommunizieren über verschiedene Broadcastnachrichten miteinander und übermitteln so sehr effizient und aktuell Performancewerte, infrastrukturelle Dienstangebote und die IP-Adressen aller bekannten QuickLink-Knoten. Das Ein- und Austreten von QuickLink-Knoten sowie Änderungen in ihrem Dienstangebot erfolgt ebenfalls über Broadcastnachrichten. Jedem QuickLink-Knoten ist somit jederzeit bekannt, welche anderen QuickLink-Knoten im Netzwerk anwesend sind, welche Infrastrukturdienste auf ihnen laufen und wie gut deren technische Leistungsfähigkeit ist. Diese Informationen werden anderen Diensten wie den Routingdiensten und dem MAS über Schnittstellen ständig zur Verfügung gestellt. Daher ist auf jedem QuickLink-Knoten sichergestellt, dass alle Informationen, die MA zur schnellen Migration auf potentere QuickLink-Knoten brauchen, immer aktuell vorhanden sind. Da QuickLink selbst wenig Ressourcen erfordert, kann es auch von schwachen Agentenplattformen (leistungsschwache Endgeräte als Hostcomputer) betrieben werden.

Auf jedem QuickLink-Knoten mit einer bestimmten Leistungsfähigkeit soll auch eine Instanz von ServiceJuggler laufen. Dies ist aber nicht zwingend notwendig. Auf schwachen Endgeräten, die keine ServiceJuggler-Instanz ausführen können oder auf Knoten, die keine Anwendungsdienste registrieren wollen (z.B. bei mobilen Hosts mit hoher Netzwerkdynamik sinnvoll), ist dann nur die Funktionalität von QuickLink vorhanden. Diese QuickLink-Knoten können somit nicht in die Verwaltungsfunktion von ServiceJuggler eingebunden werden und bekommen auch keinen Zugriff auf die Anwendungsdienstinformationen der

Region. MA auf solchen Knoten müssen daher zwingend auf eine andere Plattform migrieren, um sich eine Route planen lassen zu können.

Die QuickLink-Knoten, die eine Instanz von ServiceJuggler betreiben, können an der Wahl des Regions-Managers teilnehmen und Anwendungsdienste am Regions-Manager registrieren lassen. Die Wahl eines QuickLink-Knotens zum Regions-Manager erfolgt auf Basis der Performancedaten und der daraus gebildeten Prioritätswerte, die QuickLink liefert. Der geeignetste QuickLink-Knoten mit gestarteter ServiceJuggler-Instanz macht sich selbst zum Regions-Manager, indem er zwei zusätzliche Infrastrukturdienste startet, nämlich den

1. ServiceRegistry und
2. APLICOOVER.

Ist ein ServiceRegistry gestartet, können die ServiceJuggler-Instanzen der Region die Anwendungsdienste der Agentenplattformen registrieren, deregistrieren und nach ihnen suchen. Die Region in sich ist nun voll funktionsfähig.

Im laufenden Betrieb kann der ServiceRegistry und mit ihm die Rolle des Regions-Managers auf einen anderen QuickLink-Knoten migriert werden, wenn sich dieser aufgrund höherer Prioritätswerte als geeigneter als der bisherige herausstellt. Eine Anpassung an dynamische Veränderungen innerhalb der Region erfolgt somit automatisch und es wird sichergestellt, dass immer eine relativ stabile Plattform die Rolle des Regions-Managers übernimmt.

Der plötzliche, unvorhersehbare Ausfall eines QuickLink-Knotens wird von jeder Plattform über ihr QuickLink detektiert. In diesem Fall informieren auf jeder Plattform die QuickLink-Instanzen umgehend ihre ServiceJuggler-Instanzen. Da auch der Regions-Manager eine QuickLink- und eine ServiceJuggler-Instanz betreibt, kann er, wenn über den Ausfall einer Agentenplattform informiert wird, die Anwendungsdienste der betreffenden Plattform selbständig aus dem ServiceRegistry löschen. Über diesen Mechanismus können bei Ausfall einer Plattform auch die Anwendungdienstinformationen aktuell gehalten werden.

Der zweite spezielle Dienst des Regions-Managers, APLICOOVER, übernimmt die *Verbindung der Region zu anderen Regions-Managern im globalen Netzwerk.* Jeder Regions-Manager propagiert die kumulierten Anwendungsdienstinformationen seiner Region *(das Regionsprofil)*, verbunden mit seiner eigenen Adresse, im globalen System. Dafür wählt er nur diejenigen Anwendungsdienste aus, die eine gewisse Stabilität erwarten lassen und daher geeignet sind, im globalen System propagiert zu werden. Gleichartige Dienste auf verschiedenen Plattformen einer Region werden nur einmal global propagiert und mit der Kardinalität des Dienstes in der Region als Zusatzinformation versehen. Eine hohe Kardinalität bedeutet eine hohe Dienstredundanz und damit eine hohe Wahrscheinlichkeit, dass mindestens eine Instanz des Dienstes in der Region noch läuft, wenn ein

MA dorthin migriert ist. MA können daher aus der Kardinalität eines gesuchten Dienstes eine qualitative Auswahl geeigneter Regionen für ihre Routenplanung treffen. Durch die Stellvertreterfunktion der Regions-Manager im globalen Netzwerk müssen MA, wenn sich der anvisierte Anwendungsdienst nicht direkt auf dem Regions-Manager der fremden Region befindet, mindestens eine Migration mehr ausführen, um zur Zielplattform zu gelangen. Im Gegenzug dafür wird bei hoher Dienstredundanz in einer Region die Anzahl der im globalen System veröffentlichten Anwendungsdienste stark reduziert. Die zusätzliche Migration eines MA stellt aber qualitativ für die Erledigung seines User-Tasks keinen Nachteil dar, da es für einen MA solange keine Rolle spielt, auf welcher Plattform in einer fremden Region ein spezieller Dienst angesiedelt ist, bis sich der Agent in der fremden Region befindet[9].

Anfragen von MA nach Anwendungsdiensten, die vom regionalen ServiceRegistry nicht befriedigt werden können, werden automatisch an APLICOOVER weitergeleitet. Die Antworten APLICOOVERs werden in dessen lokalen Cache zwischengespeichert und stehen dann auch anderen Nachfragern zur Verfügung. Damit werden mit zunehmenden Anfragen und Antworten an APLICOOVER immer mehr andere Regionen direkt bekannt. Zu diesen direkt bekannten Regionen bilden sich im globalen Netzwerk kurze Links aus, die dem Small-World-Phänomen unterliegen und zu einer deutlichen Effizienzsteigerung [Kle99] bei der Suche nach Anwendungsdienstinformationen führen.

6.4 Zusammenfassung

In diesem Kapitel wurde die Architektur QuickLinkNets und dessen Einbindung in ein MAS der Stufe 2 dargestellt. Die Aufgaben der Komponenten QuickLinkNets, QuickLink, ServiceJuggler und APLICOOVER, wurden ebenso wie deren Zusammenspiel behandelt. Aus dem Zusammenspiel der Komponenten wurde der Aufbau und die Wirkungsweise des logischen Netzwerkes, welches durch QuickLinkNet aufgespannt wird, erläutert.

Die nachfolgenden Kapitel 7 bis 9 dieses Teils der Dissertation widmen sich der detaillierten Beschreibung der drei Komponenten QuickLinkNets.

[9] Vgl. dazu auch das Konzept der *Fish-Eye-View* Karten in Erfurths Dissertation [Erf04], welche MA eine detaillierte Sicht der logischen lokalen Netzwerkbereiche (dort Domains genannt) und eine reduzierte Sicht auf entfernte Bereiche bieten.

7 Ein lokales und hochdynamisches logisches Netzwerk verwalten - QuickLink

Dieses Kapitel befasst sich mit dem Infrastrukturdienst *QuickLink* als Komponente des komplexen Infrastrukturdienstes QuickLinkNet, der als Framework die Bestandteile QuickLink, ServiceJuggler und APLICOOVER in sich vereint. QuickLink dient der lokalen Vernetzung von Agentenplattformen und liefert anderen Komponenten, wie in Abbildung 7.1 dargestellt, grundsätzliche Informationen über andere Plattformen im lokalen Netzwerk.

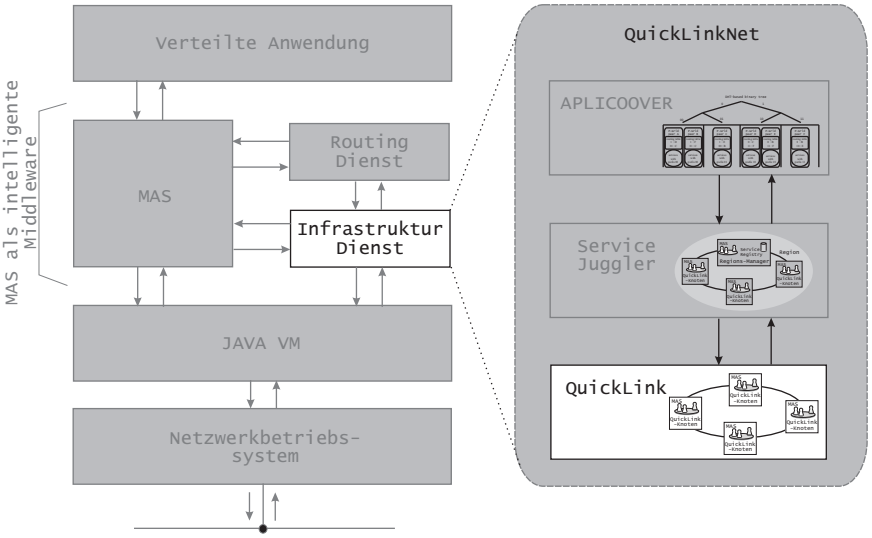

Abbildung 7.1: Der Infrastrukturdienst QuickLink im Framework QuickLinkNet

Abschnitt 7.1 gibt einen Überblick über die globalen Aufgaben QuickLinks und über die Randbedingungen, unter denen QuickLink arbeitet. Ferner werden die Teilaufgaben QuickLinks in den sich anschließenden Unterabschnitten detailliert diskutiert und analysiert.

Die darauf folgenden Abschnitte behandeln dann den Aufbau sowie die Funktionen und Wirkungsweise der prototypisch implementierten Bausteine QuickLinks. Dabei werden alle Komponenten und Themen, die mit dem Aufbau des logischen Netzwerkes befasst sind, zuerst behandelt.

Die zweite große Aufgabe QuickLinks, die Ermittlung von Leistungsparametern der unterliegenden Plattform, stellt einen vom logischen Netzwerk getrennten Problemkreis dar. Dieser wird im Abschnitt 7.1.3 analysiert. Im Abschnitt 7.6 wird die implementierte Lösung beschrieben.

7.1 Der Infrastrukturdienst QuickLink im Überblick

7.1.1 Aufgaben und Randbedingungen des logischen Netzwerkes

Die Softwarekomponente *QuickLink* des komplexen Infrastrukturdienstes *QuickLinkNet* realisiert selbst einen (einfacheren) Infrastrukturdienst, der ein lokales logisches Netzwerk zwischen Agentenplattformen aufspannt. Dies stellt die elementarste Leistung QuickLinks dar. Die Aufgaben dieses Infrastrukturdienstes wurden im Abschnitt 5.2.2 abgeleitet und sollen an dieser Stelle noch einmal kurz genannt werden:

- Hochdynamische Agentenplattformen in einem lokalen logischen Netz einbinden können,
- das Dienstangebot über Infrastrukturdienste und grundlegende Dienste eines MAS verwalten und
- die Aktualität der Dienstinformationen sicherstellen.

Die Motivation für die getrennte Betrachtung von Anwendungsdiensten gegenüber Infrastrukturdiensten und grundlegenden Diensten von MAS wurde in Abschnitt 5.2 ausführlich beschrieben. Wichtig an dieser Stelle ist, dass QuickLink daher nur Informationen über letztere Dienstarten verwaltet und diese den herrschenden Randbedingungen entsprechend behandelt. Die Randbedingungen sind:

- eine im lokalen Netzwerk begrenzte Anzahl von Knoten,
- eine geringe Anzahl von verschiedenartigen Diensten,
- eine enge Kopplung dieser Dienste an das MAS bzw. die Infrastruktur,
- daraus folgend ist die semantische Bedeutung dieser Dienste und ihre Funktionalität auf jeder Agentenplattform bekannt,
- die Aktualität dieser Dienstinformationen muss mit derselben Dynamik, wie das Ein- und Austreten hochdynamischer Agentenplattformen abgebildet werden soll, sichergestellt werden.

Für die Systemeigenschaften QuickLinks kann man aus diesen Randbedingungen ableiten, dass die Mengenskalierung der Knoten und Dienste eine untergeordnete Rolle spielt. Wichtig sind vor allem die Abbildung der hohen Netzwerkdynamik und die damit verbundene Aktualität, die daher beim Entwurf QuickLinks ausschlaggebend waren.

7.1.2 Vorbetrachtungen und Analyse zum logischen Netzwerk

7.1.2.1 Einen mobilen Host in ein Netzwerk einbinden

Die Aufgabe, eine hochdynamische mobile Agentenplattform in ein logisches Netzwerk einzubinden, ist eine komplexe Aufgabe, die QuickLink nicht allein lösen kann. Vielmehr kann und muss der Softwaredesigner einer Middleware oder einer verteilten Anwendung[1] dafür auf die Dienste der Anwendungsschicht des unterliegenden Kommunikationsprotokollstapels zurückgreifen, welches in diesem Fall die Dienste des Internetprotokollstapels sind. Diese Anwendungsdienste stellt i.d.R. das Betriebssystem zur Verfügung, im Falle einer Java-Anwendung die *Java Virtual Machine* [Jav07b] (Java VM).

Die Einbindung eines mobilen Hosts in ein Netzwerk findet auf allen Schichten des Kommunikationsprotokollstapels statt, so dass dieser Vorgang für verteilte Anwendungen weitestgehend transparent erfolgt[2]. Eine verteilte Anwendung bemerkt den Eintritt des Hostcomputers in ein Netzwerk daher frühestens, wenn der Treiber des entsprechenden Netzwerkinterfaces das Betriebssystem davon benachrichtigt und dieses wiederum diese Information höheren Anwendungsdiensten zur Verfügung gestellt hat. Glücklicherweise arbeiten die hardwarenahen Schichten so schnell, dass Verzögerungen auf der Netzwerkebene des Internetprotokollstapels im gesamten Einbindungsvorgang eine untergeordnete Rolle spielen.

Zu diesem Zeitpunkt kann eine verteilte Anwendung immer noch nicht mit dem Netzwerk kommunizieren, da die Zuteilung einer gültigen IP-Adresse noch fehlt, auf der die Kommunikation aller Anwendungsschichtdienste basiert. Die Zuteilung in einem dynamischen Netzwerk erfolgt heutzutage meist, gerade bei mobilen Hosts, über eine dynamische Adressenzuweisung, wie sie z.B. der Anwendungsschichtdienst DHCP [Dro97] leistet. Hat der mobile Host im neuen Netzwerk eine gültige IP-Adresse und Subnetzmaske[3] zugewiesen bekommen, ist der Teil der Einbindung ins Netzwerk, den das Betriebssystem des Hosts leisten muss, abgeschlossen. Frühestens ab diesem Zeitpunkt können verteilte Anwendungen, wie Middleware, MAS und QuickLink sie darstellen, ihre Aufgaben

[1] Dies stellt QuickLink aus Sicht der Kommunikationssysteme dar, vgl. dazu auch Abschnitt 2.2.4.
[2] Vgl. hierzu auch Abschnitt 2.2.
[3] Durch DHCP können auch noch andere netzwerkbezogene Parameter übermittelt und zugewiesen werden, wie z.B. die Adresse eines DNS-Servers zur Namensauflösung, der Default-Gateway usw., weitere Informationen kann man [Dro97] entnehmen.

wahrnehmen. Den zeitlichen Einbindungsvorgang des Hosts durch die Dienste des Betriebssystems haben sie bis zu diesem Zeitpunkt nicht wahrgenommen.

7.1.2.2 Eine mobile Agentenplattform in ein logisches Netzwerk einbinden

Ist ein mobiler Host, der in ein logisches Netzwerk eingebunden werden will und dazu einen entsprechenden Infrastrukturdienst wie QuickLink betreibt, erst einmal in ein neues IP-Netzwerk eingebunden, kann der Infrastrukturdienst mit seiner Arbeit beginnen. Dazu muss dieser folgende Aufgaben wahrnehmen:

- den neuen Knoten im logischen Netzwerk bekannt machen,
- ihn in die logische Struktur einbinden und
- ihn mit den für seine verteilte Anwendung, z.B. ein MAS, nötigen Informationen versorgen.

7.1.2.3 Das logische Netzwerk verwalten

Ein logisches Netzwerk aus mobilen Agentenplattformen verhält sich unter den angenommenen Randbedingungen sehr dynamisch. Jede Veränderung muss detektiert und den teilnehmenden Knoten bekannt gemacht werden. Zu den möglichen Veränderungen im Netzwerk, die verwaltet werden müssen, gehören:

- der initiale Aufbau (bootstrapping) eines neuen Netzwerkes,
- der Eintritt neuer Knoten,
- der vorhersehbare Austritt von Knoten,
- der unvorhergesehene Austritt von Knoten,
- jede Änderung von Knoteneigenschaften wie Ressourcenänderungen und
- (eventuell, je nach verwendeter Technik) der geordnete Abbau des Netzwerkes.

Diese Aufgaben können bei hochdynamischen Netzwerken zu einer hohen Netzwerklast durch Verwaltungsnachrichten führen, die bis zur Überlastung des Netzwerkes und damit zu dessen zeitweisem Ausfall führen können. Daher ist die zeitliche und mengenmäßige Skalierbarkeit der Verwaltungsnachrichten für QuickLink essentiell wichtig.

7.1.3 Vorbetrachtungen und Analyse zur Leistungsmessung

7.1.3.1 Arbeitsteilung

Die Leistungsfähigkeit einer QuickLink-Plattform bestimmt ihre Eignung für besondere Aufgaben, wie es z.B. das Hosten eines zentralen Verzeichnisdienstes (z.B. der ServiceRegistry) darstellt. Die Entscheidung darüber, welche Plattform am besten für eine bestimmte Aufgabe geeignet ist, ist eine plattformübergreifende Aufgabe und wird daher von *ServiceJuggler*, also einer Komponente außerhalb von QuickLink, getroffen. Die Eignung kann durch verschiedene Leistungsparameter bestimmt werden. Für eine Entscheidung über die Eignung sollte der am besten auf die Charakteristik einer Anwendung oder eines Dienstes passende Leistungsparameter ausgewählt werden. Ein Verzeichnisdienst soll z.B. möglichst stabil ansprechbar sein. Neben einer möglichst ressourcenstarken Plattform wäre also auch die Zuverlässigkeit, z.B. ausgedrückt durch eine möglichst lange Anwesenheitszeit im lokalen QuickLink-Netzwerk, ein wichtiger Eignungsfaktor.

QuickLink, mit seiner auf die Plattform bezogenen Sicht, ist für die Erzeugung von plattformbezogenen Leistungsparametern verantwortlich. Um eine Management- und Entscheidungskomponente wie ServiceJuggler nicht schon im Vorhinein einzuschränken, sollte eine informationenliefernde Komponente wie QuickLink möglichst viele Leistungsparameter zur Verfügung stellen. Die Auswertung der Leistungsparameter und die daraus entstehenden Entscheidungen und Aktionen müssen aber immer der entscheidenden Komponente obliegen.

7.1.3.2 Möglichkeiten zur Leistungsmessung

Um Änderungen der Leistungsfähigkeit aufgrund der dynamischen Netzwerke und dynamischen QuickLink-Knoten detektieren zu können, muss die Leistungsmessung permanent oder zumindest regelmäßig während der Lebenszeit der messenden Komponente erfolgen. Es bieten sich unter Java mehrere Möglichkeiten an, dies zu tun. Die Möglichkeiten lassen sich in zwei Prinzipien zur Ermittlung der Leistungswerte einteilen:

1. Leistungsbezogene Werte aus bestehenden Systemen verfügbar machen und auslesen (Monitoring),

2. Leistungsbezogene Werte selbst messen (Benchmarking).

Unter dem ersten Punkt lassen sich das *Simple Network Management Protocol (SNMP)* [DHW02] mit seiner standardisierten Datenstruktur *Structure of Management Information (SMI)* [RM90] und seinen standardisierten Dateneinheiten *Management Information Base (MIB)* [P+02] ebenso wie die *Java Management Extensions (JMX)* [Sun07a] und die Methoden zum Erhalt von Informationen

der Laufzeitumgebung einer Java-Anwendung[4] subsumieren. Neben den direkt durch Java ansprechbaren Werkzeugen ist es auch möglich, Java-fremde, auf den Programmiersprachen C oder C++ basierende Werkzeuge zu verwenden, die ihre Werte über das *Java Native Interface (JNI)* [Sun03] an QuickLink weitergeben können.

Die zweite Möglichkeit, nämlich Leistungswerte selbst zu messen, manifestiert sich in *Benchmark* genannten Programmen, welche standardisierte Messprogramme zur Ermittlung der Rechnerleistung [RP99] darstellen. Benchmarks messen die Leistung eines Computersystems intrusiv, d.h. sie erzeugen selbst Last in verschiedenen Formen. Dazu werden im Prinzip verschiedene, vorher im Benchmark definierte Aufgaben vom Rechner ausgeführt und die Zeit für die erfolgreiche Bewältigung der Gesamtaufgabe gemessen. Je schneller ein Computersystem die vom Benchmark gestellte Aufgabe löst, desto höher ist seine Leistungsfähigkeit.

Benchmarkergebnisse dienen der Vergleichbarkeit von Rechensystemen auf sehr hohem Niveau und mit sehr hoher Genauigkeit. Die Kriterien für Benchmarks sind daher standardisiert[5] und werden streng überwacht. Im Bereich des wissenschaftlichen Hochleistungscomputing wurden u.a. von der *EPCC* [EPC07a] Anstrengungen unternommen, hochperformante Anwendungen in Java zu implementieren und auch entsprechende Benchmarks zur Evaluation der Java-Umgebungen anzubieten. Ein bekannter und verbreiteter Benchmark ist die Java Grande Forum Benchmark Suite [EPC07b] der EPCC.

7.1.3.3 Bewertung der Möglichkeiten zur Leistungsmessung

Die Möglichkeiten zum Monitoring verschiedener Leistungsparameter sind vielfältig, aber nicht alle sind für QuickLink geeignet. SNMP erfordert eine spezielle Clientsoftware und einen laufenden SNMP-Dienst, dessen Vorhandensein nicht unbedingt vorausgesetzt werden kann. Daher erscheint SNMP für die Leistungsmessung in QuickLink nicht geeignet. Die Einbindung externer, auf C oder C++ basierender Programme über JNI führt zu einer Abhängigkeit vom Betriebssystem und zur Abkehr vom Java-Prinzip *"Write once, run anywhere" (WORA)*. Diese Lösung erscheint also noch weniger geeignet als SNMP.

Die direkt aufrufbaren und die über JMX verfügbaren Methoden der SUN Java VM sind hingegen in jedem Falle in der Standard-*Java Laufzeitumgebung (JRE)*

[4] Gemeint sind Informationen über Speicher, Speicherauslastung und Prozessoranzahl, wie sie z.B. die Java VM der Firma SUN über die Methoden des Runtime-Objektes [Sun04a] einer Java-Anwendung zur Verfügung stellt.

[5] Professionelle Benchmarks zur Bewertung von Rechensystemen werden heute von unabhängigen Organisationen wie der *Standard Performance Evaluation Corporation (SPEC)* [Sta07] und der *Transaction Processing Performance Council (TPC)* [Tra07] entwickelt und freigegeben. Leider sind diese Benchmarks auch kostenpflichtig.

verfügbar [Sun04b] und bedürfen keiner besonderen Implementierung. Ihre Benutzung bietet sich daher an, um in Bezug auf standardisierte Implementierung und Zuverlässigkeit gute Messergebnisse zu erzielen. Allerdings bieten diese Methoden keine aktuellen, auf die Rechenleistung bezogenen Leistungswerte an.

Für die Messung der Rechenleistung bleiben also nur noch Benchmarks übrig. Klassische Benchmarks erfordern aber Umgebungseigenschaften, die nicht mit einem im praktischem Betrieb eingesetzten Computer vereinbar sind:

- Ein sauberer Computer, auf dem keine zusätzlichen Dienste und Anwendungen laufen, sondern nur der Benchmark ausgeführt wird.
- Möglichst lange Messzeiten (mehrere Minuten bis Stunden), um ein möglichst genaues Ergebnis zu erzielen.

Unter diesen Bedingungen können sehr genaue und vor allem gut vergleichbare Ergebnisse erzielt werden. Leider sind diese Einsatzbedingungen nur auf speziell präparierten Computern anzutreffen und für das Einsatzszenario von QuickLink nicht geeignet. QuickLink bietet im Einsatz hingegen folgende Eigenschaften:

- Ein schwankend ausgelasteter Computer.
- Eine nicht vorhersagbare Anzahl laufender Dienste.

Zusätzlich wird von einer leistungsmessenden Komponente Folgendes verlangt:

- Die laufenden Dienste sollen trotz intrusiver Messmethoden möglichst wenig von der Leistungsmessung beeinflusst werden.
- Deshalb muss die Laufzeit der Messungen im Vergleich zu einem Benchmark sehr kurz und die Belastung möglichst gering sein.
- Die Messung muss in regelmäßigen Abständen wiederholt werden, um die unterschiedlichen Auslastungen des Knotens detektieren zu können.

Für diese Umgebungsbedingungen sind klassische Benchmarks nicht gemacht und damit auch nicht einsetzbar. Vielmehr muss eine eigene, abgespeckte Leistungsmessung, die sich eventuell an die Methoden von Benchmarks anlehnt, ersonnen werden. Sie darf nur wenig Laufzeit erfordern und wenig Rechenlast erzeugen, um die Selbstbeeinflussung eines QuickLink-Knotens so gering wie möglich zu halten. Trotzdem muss gewährleistet sein, dass Plattformen anhand der gemessenen Leistungswerte wenigstens grob in Rechnerklassen eingeteilt und miteinander verglichen werden können.

7.1.4 Aufbau und Architektur des Infrastrukturdienstes QuickLink

Nach den Vorbetrachtungen zu den Aufgaben des Infrastrukturdienstes QuickLink wird nun der architekturelle Aufbau des Dienstes betrachtet. Der Infrastrukturdienst QuickLink ist in mehrere Softwarekomponenten gegliedert, welche jeweils eine abgeschlossene Funktionalität in sich einkapseln. Dadurch ist der Austausch wesentlicher Funktionalitäten einfach möglich, falls sich ein Bedarf an funktionalen Verbesserungen oder Veränderungen ergeben sollte. Dabei sind die einzelnen Komponenten als nebenläufige Prozesse (Java Threads) implementiert, die miteinander kommunizieren. Die Komponenten und ihr Zusammenspiel sind in Abbildung 7.2 zu sehen. Als wesentliche funktionale Bestandteile des Infrastrukturdienstes QuickLink kristallisieren sich die Softwarekomponenten

- *QuickLink*,
- der *CycleManager*,
- der *Kicker*,
- der *PerformanceAnalyser* und
- die *Interfaces*

heraus.

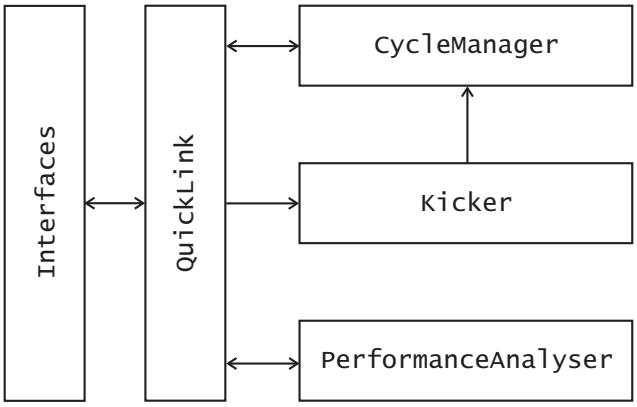

Abbildung 7.2: Struktur der Softwarekomponenten des Infrastrukturdienstes QuickLink

Der PerformanceAnalyser leistet die Leistungsmessung auf dem QuickLink-Knoten und nimmt daher in dieser Aufstellung eine Sonderstellung ein, da seine Funktionalität für die Bildung des lokalen logischen Netzwerkes nicht zwingend benötigt wird. Bei der detaillierten Vorstellung der einzelnen Komponenten

QuickLinks in den folgenden Abschnitten werden deshalb die Komponenten, die direkt für das logische Netzwerk zuständig sind, zuerst und der Performance-Analyser zuletzt in Abschnitt 7.6 behandelt.

7.2 Die Softwarekomponente QuickLink

Die zentrale funktionale Softwarekomponente des *Dienstes QuickLink* stellt die gleichnamige *Komponente QuickLink* dar. Sie übernimmt die Manager-Funktion für die einzelnen funktionalen Bestandteile des Dienstes. Die Komponente QuickLink koordiniert folgende Aufgaben:

- Start, Initialisierung und Terminierung des CycleManagers, des Kickers und des PerformanceAnalysers,
- die Koordination dieser Komponenten,
- das Triggern der Leistungsmessung,
- das Sammeln, die Aufbereitung und die Bereitstellung aller Daten, die aus dem logischen Netzwerk und dem PerformanceAnalyser geliefert werden und
- die Kommunikation nach außen über die Interfaces.

Zur Parametrisierung des Dienstes QuickLink von außen dient eine Konfigurationsdatei, die beim Start ausgelesen und mit deren Werten die Softwarekomponente QuickLink initialisiert wird. QuickLink startet dann seinerseits die anderen Komponenten mit ihren initialen Parametern. Alle QuickLink-Knoten eines logischen lokalen Netzwerkes müssen zwingend mit einer Konfigurationsdatei gleichen Inhalts initialisiert werden, da die übergebenen Parameter die innere Struktur der verwendeten Datenspeicher, der Nachrichten und auch der zeitlichen Abläufe festlegen. Es muss sichergestellt sein, dass alle QuickLink-Knoten die gleichen Daten- und Nachrichtenformate und die gleichen zeitlichen Parameter verwenden, um sich synchronisieren zu können.

Die aktuellen netzwerkbezogenen Daten, die QuickLink vom CycleManager bezieht, bereitet es auf, um sie in geeigneten Datenformaten über die Interfaces zur Verfügung zu stellen. Dabei werden die Daten in verschiedenen Formen redundant angeboten, um dem Anforderungsniveau der jeweiligen Kommunikationspartner, wie z.B. MAS oder Routingdiensten, gerecht zu werden. So werden z.B. Leistungsparameter sowohl im Datentyp *Long* für eine genaue Auswertung durch beliebige Anwendungen angeboten als auch im Datentyp *Byte*, der als Prioritätswert zur QuickLinkNet-internen Steuerung genutzt wird. Zur Umrechnung werden die Werte der Datentypen durch Division mit einem Vielfachen von

zehn in Werte umgerechnet, die sich in den Grenzen der Zahlenwerte des Datentyps Byte abbilden lassen. So wird z.B. der maximale Speicher einer Plattform als Long-Wert in KByte und als Byte-Wert in jeweils 10 MByte angegeben. Die Aufenthaltszeit im Netzwerk steht als Long-Wert in Sekunden oder Byte-Wert in Minuten zur Verfügung. Die Rechenleistung kann ebenfalls als ausführlicher Long-Wert oder als Byte-Wert abgerufen werden. Zur Ausnutzung des gesamten Zahlenbereichs des Datentyps Byte erfolgt bei der Aufenthaltszeit im Netzwerk und bei der Rechenleistung zusätzlich eine Zahlenbereichsverschiebung um -128, so dass die Skala bei -128 beginnt und bei 127 endet. Eine vollständige Übersicht über die zur Verfügung gestellten Werte findet man im Abschnitt 7.5, in dem die Interfaces behandelt werden. Die genaue Umrechnung der Werte wird im Anhang A.2 erläutert. Die Weiterverwendung der Leistungswerte im Rahmen von QuickLinkNet wird im Kapitel 8 behandelt.

Wird QuickLink beendet, terminiert es zuerst die von ihm gestarteten Softwarekomponenten, bevor es sich selbst beendet.

7.3 Die Softwarekomponente CycleManager

Die Softwarekomponente CycleManager beinhaltet den größten Funktionalitätsumfang des Dienstes QuickLink. Er steuert die gesamte Logik zur Verwaltung des lokalen logischen Netzwerkes und zur Kommunikation mit diesem.

7.3.1 Aufbau und Realisierung

7.3.1.1 Architektur und Bestandteile

Der CycleManager ist als Java-Klasse implementiert, die alle ihre funktionalen Einheiten als interne Klassen einkapselt. Die funktionalen Einheiten ergeben sich aus den Aufgaben, die der CycleManager zu erfüllen hat. Für jede funktionale Einheit beinhaltet der CycleManager eine interne Klasse, die den Aufgaben entsprechend mehrfach instanziiert werden. Die Aufgaben und die zugehörigen Klassen sind in folgender Aufzählung gegenübergestellt:

- Datenhaltung (QuickLink-Knoten)- *QuickLink-List*,
- Datenhaltung (Priortäten der QuickLink-Knoten)- *Priority-List*,
- grundsätzliche Netzwerkkommunikation - *UDP-Socket*,
- Empfang von Netzwerknachrichten - *Receiver*,
- zeitliche Synchronisation - *Timer*.

Die hauptsächlichen Funktionen des CycleManagers, die das Management der Informationen, die Aktualisierung der Daten und das Kommunizieren mit dem Netzwerk beinhaltet, ist im Haupt-Thread des CycleManagers implementiert. Diese Funktionen werden im Abschnitt 7.3.2 beschrieben.

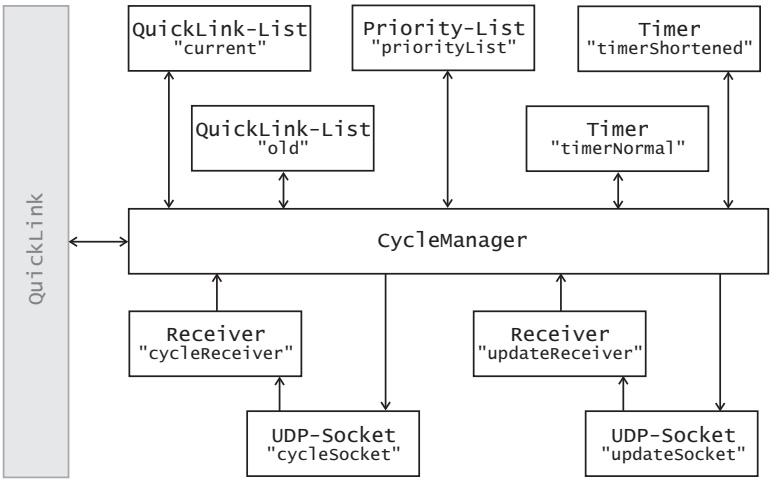

Abbildung 7.3: Die funktionale Struktur des CycleManagers

Der CycleManager-Thread benötigt für die Realisierung seiner eigenen Funktion mehrere verschiedene Instanzen seiner Unterklassen. Ein funktionsfähiger CycleManager benötigt, wie zur Verdeutlichung in Abbildung 7.3 dargestellt, folgende Objekte:

QuickLink-Lists: Der CycleManager hält seine Daten in einer eigenen Datenstruktur, der *QuickLink-List*, welche zweimal instanziiert wird. Eine QuickLink-List namens *"Old"* repräsentiert immer die Daten, die aktuell im gesamten QuickLink verwendet werden. Wird eine neue Zyklusnachricht empfangen, so werden deren Daten in der QuickLink-List *"Current"* gespeichert. Im nachfolgenden Verarbeitungsprozess des CycleManagers werden die beiden QuickLink-Lists miteinander verglichen, auf Statusänderungen überprüft und entsprechende Reaktionen ausgelöst.

Priority-List: Die *Priority-List* ist eine Datenstruktur zur Speicherung der Prioritätswerte aller QuickLink-Knoten eines QuickLink-Netzwerkes. Im Gegensatz zur QuickLink-List wird sie innerhalb des CycleManagers nur einmal als *"priorityList"* instanziiert. Sie wird mit den Prioritätsdaten aus den Headern der empfangenen Zyklusnachrichten gespeist. Diese Daten werden dem im Kapitel 8 behandelten Infrastrukturdienst *ServiceJuggler* zur Verfügung gestellt und dienen als Ausgangsbasis für die Wahl des Regions-Managers.

UDP-Sockets: Der CycleManager verwendet zwei verschiedene Ports, um mit dem Netzwerk per Broadcast zu kommunizieren. Die verschiedenen Ports sind deshalb nötig, weil die verschiedenen Arten von Nachrichten, die Zyklusnachrichten und die Updatenachrichten, zeitgleich eintreffen können. Für jeden Port muss daher ein UDP-Socket eröffnet werden. Entsprechend ihrer Funktion heißen die UDP-Sockets *"cycleSocket"* und *"updateSocket"*.

Receiver: Um auf den UDP-Sockets Nachrichten empfangen zu können, benötigt der CycleManager jeweils einen Receiver, der unabhängig vom übrigen Geschehen ständig auf Nachrichten wartet. Die Receiver sind daher als Java-Threads implementiert und heißen entsprechend der zugehörigen UDP-Sockets *"cycleReceiver"* und *"updateReceiver"*. Das Senden eigener Nachrichten erledigt der CycleManager direkt auf den UDP-Sockets.

Timer: Die Timer dienen der Synchronisation der Empfangsvorgänge für Zyklusnachrichten. Sie messen die Zeit, die seit der letzten empfangenen Zyklusnachricht vergangen ist. Die zu messenden Zeiten gelten für alle QuickLink-Knoten gleichzeitig. Die Zeit nach dem Empfang der letzten Zyklusnachricht wird vom *"timerNormal"* gemessen. Er bestimmt somit den Zyklustakt der zu versendenden Zyklusnachrichten in einem lokalen logischen QuickLink-Netzwerk. Wird nach Ablauf der Zeit t_{normal} des *timerNormal* keine Nachricht empfangen, wartet der CycleManager noch eine Karenzzeit $t_{waiting}$ mit $t_{waiting} < t_{normal}$ ab, um eine etwas verspätet eintreffende Nachricht doch noch berücksichtigen zu können. Ist bis dahin keine Nachricht empfangen worden, wird der Timer *"timerShortened"* mit $t_{shortened}$ mit $t_{shortened} = t_{normal} - t_{waiting}$ gestartet, um die Synchronität zum Zyklustakt aufrecht zu erhalten.

7.3.1.2 QuickLink-List

Die QuickLink-List ist die zentrale Datenstruktur im CycleManager. Sie speichert die kumulierten, dienstbezogenen Informationen eines QuickLink-Knotens bezüglich aller im logischen QuickLink-Netzwerk bekannten QuickLink-Knoten, inklusive der Daten des eigenen Knotens. Jedem bekannten QuickLink-Knoten ist in einer QuickLink-List ein Datensatz *(row)* zugeordnet. Dieser besteht aus seiner IP-Adresse und Informationen über die aktuell von ihm ausgeführten Infrastrukturdienste und die grundsätzlichen Dienste seines MAS. Abbildung 7.4 zeigt den Aufbau eines QuickLink-List-Datensatzes.

Der erste Teil besteht aus der IP-Adresse, deren Länge bei der Initialisierung des QuickLink-Dienstes festgelegt werden muss. Dadurch ist eine Verwendung von IPv4- oder IPv6-Adressen möglich. In der Abbildung wurde eine IPv4 Adresse gewählt, die durch vier Bytes repräsentiert wird. Der zweite Teil des

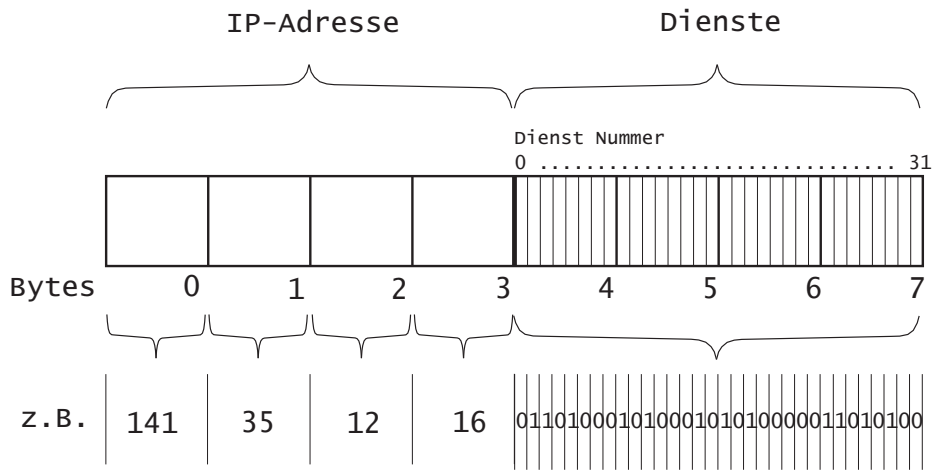

Abbildung 7.4: Der beispielhafte Datensatz einer QuickLink-List: Jeder Datensatz repräsentiert die IP-Adresse und die Dienste eines QuickLink-Knotens im lokalen logischen Netzwerk

Datensatzes besteht aus einer Anzahl von Bytes, welche ebenfalls bei der Initialisierung des QuickLink-Dienstes festgelegt werden muss. Sie repräsentieren die Dienste und zwar bitweise als logische Boolean-Werte. Jedem Dienst ist ein Bit in einem Byte zugeordnet, wobei eine 1 für ein logisches *true* und eine 0 für ein logisches *false* verwendet wird. Mit dem logischen Wert wird ausgedrückt, ob der betreffende Dienst auf dem QuickLink-Knoten ausgeführt wird oder nicht. Die Zuordnung und Reihenfolge der Dienste wird ebenfalls über die Konfigurationsdatei von QuickLink festgelegt. Sie muss auf allen QuickLink-Knoten gleich sein und ist daher explizit auf allen Knoten bekannt. Die Anzahl der für die Dienstrepräsentation benutzten Bytes und deren Bedeutung bzw. Übersetzung in Dienstnamen ist ebenfalls über die Konfigurationsdatei einstellbar. Durch die bitweise Repräsentation von Diensten und die auf Bytes festzulegende Anzahl von repräsentierten Diensten ergibt sich immer ein Vielfaches von acht zur Darstellung von Diensten im QuickLink-Netzwerk. In Abbildung 7.4 wurden beispielhaft vier Bytes zur Repräsentation von 32 Diensten verwendet, was der von uns erwarteten maximalen Anzahl von Infrastrukturdiensten und grundsätzlichen Diensten eines MAS genügen sollte.

Die QuickLink-List selbst besteht, wie in Abbildung 7.5 illustriert, aus einem Header und einer Liste aus Datensätzen der bekannten QuickLink-Knoten. Dabei sind im Header einer QuickLink-List immer Informationen über die QuickLink-List selbst und über den QuickLink-Knoten, der die QuickLink-List besitzt, gespeichert. Die Headerfelder bestehen aus jeweils einem Byte und repräsentieren folgende Werte:

- Feld 0: Anzahl der in der QuickLink-List gespeicherten QuickLink-Knoten (entspricht gleichzeitig der Anzahl der im lokalen logischen QuickLink-Netzwerk anwesenden QuickLink-Knoten),
- Feld 1: Aktuelle Rechenleistung des sendenden Hostrechners, repräsentiert als Prioritätswert durch einen Bytewert,
- Feld 2: Maximaler, durch den Hostrechner zugewiesener Speicher der Java VM in jeweils 10 MByte, repräsentiert als Prioritätswert durch einen Bytewert,
- Feld 3: Speicherauslastung der Java VM, bezogen auf den maximal zugewiesenen Speicher, in Prozent, repräsentiert als Prioritätswert durch einen Bytewert,
- Feld 4: Aktuelle Anwesenheitszeit des QuickLink-Knotens im lokalen logischen Netzwerk in Minuten, repräsentiert als Prioritätswert durch einen Bytewert.

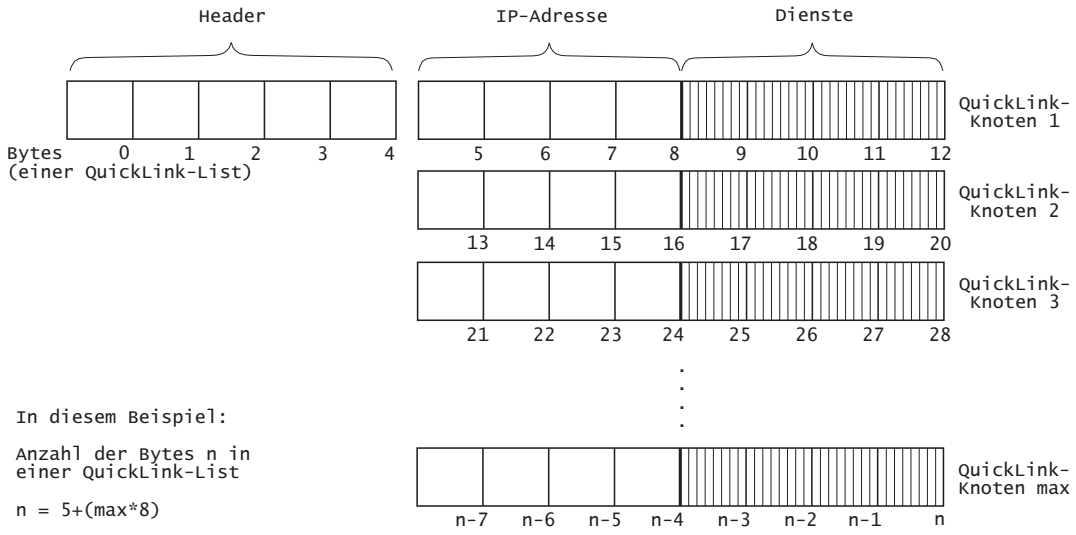

Abbildung 7.5: Die beispielhafte Struktur einer QuickLink-List: Der Header besteht aus fünf Byte-Werten, daran schließen sich die Datensätze der QuickLink-Knoten an

An den Header schließen sich die Datensätze der bekannten QuickLink-Knoten in einer vorgegebenen Reihenfolge an und bilden eine geordnete Liste. Die Reihenfolge der Datensätze wird durch die IP-Adressen der QuickLink-Knoten bestimmt. Die Liste der QuickLink-Knoten beginnt mit dem Knoten mit der klein-

sten IP-Adresse und endet mit dem Knoten mit der größten Adresse. Bei der Erstellung der Reihenfolge wird byteweise vorgegangen, d.h. immer das höchstwertige Byte (das erste Byte) der IP-Adresse wird zuerst verglichen. Haben diese Bytes den selben Zahlenwert, werden die nächst niedrigeren Bytes zweier Adressen verglichen usw., womit sichergestellt ist, dass immer die richtige Reihenfolge eingehalten wird. Dies ist wichtig, da bei der Verwaltung des QuickLink-Netzwerkes durch diese Reihefolge auch bestimmt wird, welcher Knoten als nächstes an der Reihe ist, eine Zyklusnachricht zu senden. Da die Gesamtgröße der QuickLink-List und damit die maximale Anzahl der verwaltbaren QuickLink-Knoten durch die Konfigurationsdatei im Vorhinein festgelegt wird, existieren leere Datensätze, wenn die maximal verwaltbare Anzahl von QuickLink-Knoten nicht ausgenutzt wird. Dies ist im normalen Betrieb der Regelfall und nicht die Ausnahme. Die leeren Datensätze werden mit Nullen aufgefüllt.

Die Implementierung einer QuickLink-List besitzt alle Funktionen, um ihre Einträge gezielt manipulieren zu können. Der CycleManager besitzt wiederum alle nötigen Funktionen, um die zwei QuickLink-Lists "Old" und "Current" miteinander zu vergleichen und aus den Unterschieden die entsprechenden Schlüsse über den Zustand des QuickLink-Netzwerkes zu ziehen und entsprechend zu agieren.

Die Struktur der QuickLink-List hängt eng mit der Form und den Inhalten der Nachrichten zusammen, die im QuickLink-Netzwerk verwendet und im Abschnitt 7.3.1.4 beschrieben werden. Zuvor wird aber noch die zweite Datenstruktur des CycleManagers, die Priority-List, beschrieben.

7.3.1.3 Priority-List

Die Priority-List beinhaltet im Gegensatz zur QuickLink-List nur die leistungsbezogenen Daten aller bekannten Knoten des QuickLink-Netzwerkes in Form von Prioritätswerten. Sie ist aus Effizienzgründen getrennt von den QuickLink-Lists implementiert. Die Ursachen dafür liegen in den zeitlichen Unterschieden zwischen den Aktualisierungsraten der Leistungsparameter[6] und der Dienstinformationen, welche sich auch in den Inhalten der QuickLink-Nachrichtentypen widerspiegeln[7].

Die Priority-List stellt eine geordnete Liste von QuickLink-Knoten dar und ähnelt somit vom Aufbau her, bis auf den Header, einer QuickLink-List. Jeder Listeneintrag besteht aus der IP-Adresse (im *Byte*-Format) und den Prioritätswerten eines QuickLink-Knotens (ebenfalls im *Byte*-Format). Die Werte für die Länge der IP-Adresse und Anzahl der Prioritätswerte werden ebenfalls, wie bei

[6] Die Aktualisierungsraten des leistungsmessenden Moduls PerformanceAnalyser werden im Abschnitt 7.6.2 behandelt.

[7] Die zeitlichen Abläufe im QuickLink-Netzwerk und die Inhalte der QuickLink-Nachrichten werden im nächsten Abschnitt 7.3.1.4 behandelt.

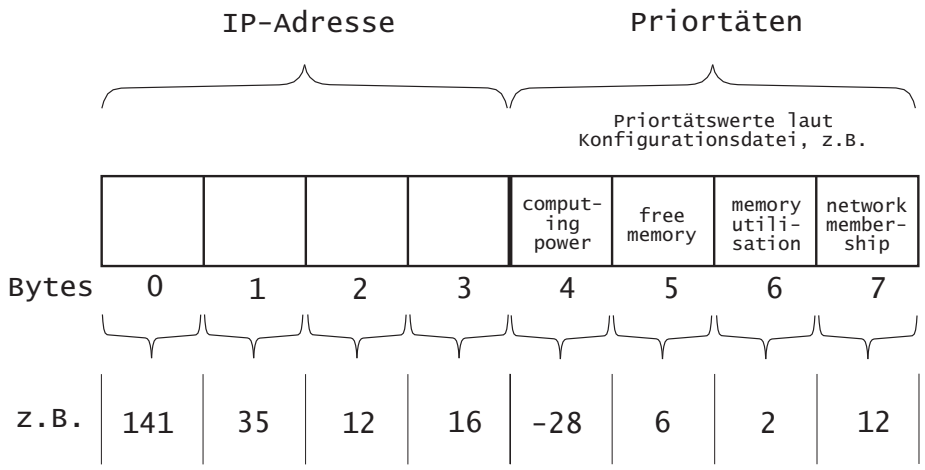

Abbildung 7.6: Der beispielhafte Datensatz einer Priority-List: Jeder Datensatz repräsentiert die IP-Adresse und die Prioritäten eines QuickLink-Knotens im lokalen logischen Netzwerk

der QuickLink-List, aus der Konfigurationsdatei QuickLinks entnommen. Abbildung 7.6 zeigt einen beispielhaften Datensatz einer Priority-List. Die verwendeten Prioritäten sind gleichartig und in der gleichen Reihenfolge angeordnet wie im Header der QuickLink-List[8]:

- Feld 4: Rechenleistung,
- Feld 5: Maximaler, durch den Hostrechner zugewiesener Speicher der Java VM in jeweils 10 MByte,
- Feld 6: Speicherauslastung der Java VM, bezogen auf den maximal zugewiesenen Speicher, in Prozent, und
- Feld 7: Aktuelle Anwesenheitszeit des QuickLink-Knotens im lokalen logischen Netzwerk in Minuten.

Die maximale Länge einer Priority-List entspricht der in der Konfigurationsdatei festgelegten, maximalen Anzahl verwaltbarer QuickLink-Knoten im QuickLink-Netzwerk. Die Reihenfolge der Einträge der Priority-List entspricht dem numerischen Wert der IP-Adresse und wird, wie im vorherigen Abschnitt 7.3.1.2 beschrieben, analog zur QuickLink-List berechnet. In Abbildung 7.7 ist der sich aus diesen Bedingungen ergebende Aufbau einer Priority-List beispielhaft illustriert.

[8] Vgl. dazu die Inhalte des QuickLink-List Headers auf Seite 115.

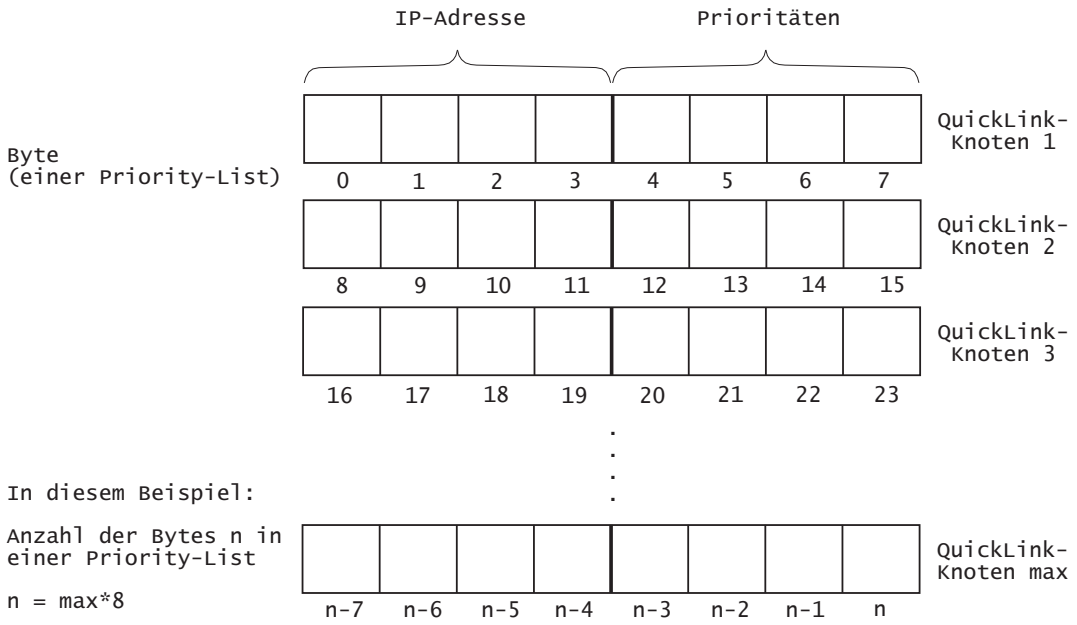

Abbildung 7.7: Die beispielhafte Struktur einer Priority-List

Die Implementierung der Priority-List besitzt alle notwendigen Funktionen, um neue QuickLink-Knoten in die Liste einfügen, nicht mehr existierende Knoten löschen und bei sich ändernden Werten existierende Knoten aktualisieren zu können. Ferner existieren zwei spezielle Methoden[9] zum Auslesen der Liste, die speziell für ServiceJuggler implementiert worden sind.

7.3.1.4 QuickLink-Nachrichten

Nachrichtenarten Zur Kommunikation verwendet der CycleManager drei Arten von Broadcastnachrichten, nämlich

- Zyklusnachrichten (cycle messages),
- Beitrittsnachrichten (join messages) und
- Aktualisierungsnachrichten (update messages).

Der Inhalt einer Zyklusnachricht entspricht einer kompletten QuickLink-List, wie sie in Abbildung 7.5 dargestellt ist. Zur Bildung der Nachricht wird die QuickLink-List serialisiert. Den beispielhaften Aufbau einer Zyklusnachricht illustriert Abbildung 7.8.

[9] Diese Methoden werden über das Interface *Priorities* angeboten und sind im Abschnitt 7.5.1 auf Seite 133 beschrieben.

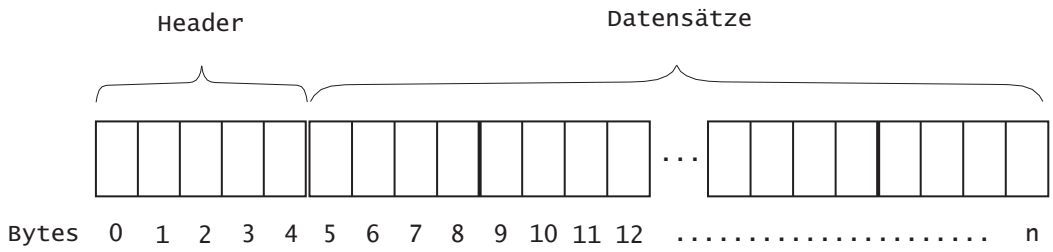

Abbildung 7.8: Der beispielhafte Aufbau einer Zyklusnachricht

Beitritts- und Aktualisierungsnachrichten gleichen sich im Aufbau, da ein Beitritt im QuickLink-Netzwerk nur einen speziellen Fall einer Aktualisierung darstellt. Der Unterschied zwischen ihnen besteht in der Art der Verarbeitung durch den CycleManager. Eine Aktualisierungsnachricht besteht, wie in Abbildung 7.9 veranschaulicht, aus dem Header der aktuellen QuickLink-List und einem einzigen QuickLink-List-Datensatz, nämlich dem des sendenden QuickLink-Knotens.

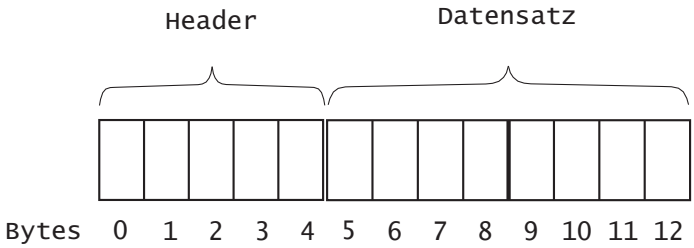

Abbildung 7.9: Der beispielhafte Aufbau einer Aktualisierungsnachricht

Zusammenhang zwischen Nachrichten und Datenstrukturen Mit dem Empfang einer Zyklusnachricht erhält jeder QuickLink-Knoten die Informationen bezüglich der laufenden grundsätzlichen Dienste eines MAS und der Infrastrukturdienste aller bekannten QuickLink-Knoten im QuickLink-Netzwerk. Dies erfolgt regelmäßig im Intervall mit einer Länge von t_{cycle}, so dass auch die QuickLink-Lists in diesem Intervall, auch Zyklustakt genannt, aktualisiert werden.

Während mit jeder Zyklusnachricht die Dienstinformationen aller QuickLink-Knoten verbreitet und aktualisiert werden, trifft dies für die Prioritäten nicht zu. Mit jeder Zyklusnachricht werden immer nur die aktuellen Prioritäten des sendenden QuickLink-Knotens verbreitet, so dass sich die Priority-List bei n Knoten frühestens nach einem kompletten Sendedurchlauf $n * t_{cycle}$ aktualisiert hat. Eine höhere Aktualisierungsfrequenz ist aber auch nicht nötig, da die Prioritäten ohnehin nur in größeren Zeitabständen als dem Zyklustakt gemessen werden und somit eine ständige Aktualisierung im Zyklustakt zu einer redundanten Übertragung immer gleicher Prioritätswerte führen würde. Es ist den QuickLink-Knoten

allerdings ebenso möglich, bei der Änderung einer oder mehrerer Prioritäten eine Aktualisierungsnachricht an das QuickLink-Netzwerk zu senden. In diesem Falle kann die Zeit zur Aktualisierung der Prioritäten auf nur t_{cycle} verringert werden.

7.3.1.5 Netzwerkkommunikation

Paradigma Aufgrund der abbildbaren hohen Netzwerkdynamik und der sich daraus ergebenden Zeitschranken steht in QuickLink eine schnelle Kommunikation mit allen beteiligten Netzwerkknoten bei gleichzeitig niedriger Netzwerklast im Mittelpunkt. Wir haben uns daher bei der Realisierung für eine Broadcastkommunikation entschieden, die auf jeder Ebene des Kommunikationsprotokollstapels effizient möglich[10] ist. Im Gegensatz zu vielen anderen Systemen zur Verwaltung lokaler logischer Netzwerke, bei denen Broadcast nur bei der Initialisierung zur Propagation von Informationen verwendet und danach auf TCP-Unicast-Verbindungen umgestiegen wird, verwendet QuickLink ausschließlich Broadcastnachrichten. Mit einer solchen Broadcastnachricht werden alle Knoten im Netzwerk gleichzeitig [11] erreicht, und dies mit nur einer gesendeten Nachricht.

Ports und Sockets Der CycleManager, der die Broadcastkommunikation innerhalb QuickLinks übernimmt, verwendet die im vorherigen Abschnitt vorgestellten Nachrichtentypen. Zum Senden und Empfangen von Zyklusnachrichten verwendet der CycleManager einen bestimmten Port, mit welchem er den UDP-Socket *cycleSocket* initialisiert. Ebenso wird mit den Beitritts- und Aktualisierungsnachrichten verfahren, die sich einen anderen UDP-Socket, den *updateSocket*, teilen. Beide verwendeten Ports müssen in der Konfigurationsdatei QuickLinks angegeben werden und für alle QuickLink-Knoten gelten. Auf den Sockets wird simultan empfangen und gesendet.

Nachrichtengrößen Broadcastnachrichten können im Protokollstapel des Internets nur über das UDP-Protokoll versendet werden. Für den vollständigen Empfang einer Nachricht, die über UDP empfangen wird, muss aber deren Größe vorher explizit bekannt sein. Dies ist notwendig, damit ein UDP-Socket weiß, nach wievielen empfangenen Bytes er die Nachricht vollständig empfangen hat und er sie weitergeben kann. Die Größe einer Zyklusnachricht ist daher genau auf die Anzahl der Bytes einer serialisierten QuickLink-List, wie in Abbildung 7.8 dargestellt, und die Größe der Aktualisierungs- und Beitrittsnachrichten auf die

[10] Vergleiche dazu auch Abschnitt 2.3.

[11] Wegen der Laufzeit der Nachricht auf dem lokalen Netzwerkmedium erreicht diese nicht alle Teilnehmer wirklich gleichzeitig. Die auf dem Netzwerkmedium auftretenden Laufzeitunterschiede können aber in Bezug auf die zeitlichen Abläufe und die Behandlung der Broadcastnachrichten in QuickLink vernachlässigt werden, so dass die Broadcastnachrichten im QuickLink-Netzwerk bei allen QuickLink-Knoten quasi gleichzeitig eintreffen.

Größe *QuickLink-List-Header + eine Datensatzgröße*, wie in Abbildung 7.9 angegeben, festgelegt. Die Größe der Nachrichten ergibt sich daher implizit aus den Festlegungen der inneren Strukturen QuickLinks in der Konfigurationsdatei. Eine sinnvolle Größe der Nachrichten ist aber nicht ausschließlich von den Anforderungen QuickLinks abhängig, sondern auch von den Leistungsparametern des unterliegenden Netzwerkes. So wäre es natürlich für eine Nachricht, die über das inhärent unzuverlässige UDP-Protokoll versendet wird, immer gut, wenn sie innerhalb der maximalen Paketgrößen der unterliegenden Netzwerke versendet werden könnte. Einige Überlegungen über eine sinnvolle Wahl der inneren Strukturen QuickLinks werden im Teil III im Abschnitt 11.2.1.3 dieser Dissertation bei der Evaluation QuickLinkNets angestellt.

Netzwerklast Die Datenstrukturen QuickLinks legen die Nachrichtengrößen fest und haben daher einen direkten Einfluss auf die Netzwerklast, die durch die Verwaltung des QuickLink-Netzwerkes entstehen. Die konkrete entstehende Belastung $L_{maintenance}$ ergibt sich mit Nachrichtengröße $N_{maintenance}$ und Sendefrequenz F aus

$$L_{maintenance} = N_{maintenance} * F \ ,$$

wobei sich die Nachrichtengröße $N_{maintenance}$ für eine Zyklusnachricht mit einer Headerlänge l_{header}, einer IP-Adressenlänge $l_{address}$, einer Dienstfeldlänge $l_{services}$ und einer maximalen verwaltbaren Anzahl von QuickLink-Knoten pro logischem lokalen Netzwerk max aus

$$N_{maintenance} = l_{header} + (max * (l_{address} + l_{services}))$$

ergibt. Da alle die Nachrichtengröße bestimmenden Parameter einschließlich der maximal zu verwaltenden Anzahl von QuickLink-Knoten und auch die Sendefrequenz der Zyklusnachrichten schon vor dem Start eines QuickLink-Netzwerkes feststehen, bleibt die Netzwerkbelastung zur Verwaltung bezogen auf die Anzahl der teilnehmenden Knoten konstant.

Im Gegensatz zur deterministisch anfallenden Netzwerklast für die reguläre Netzwerkverwaltung kann die Last $L_{dynamic}$, die durch die Beitritts- und Aktualisierungsnachrichten ausgelöst wird, nicht vorhergesagt werden. Dies liegt an der unvorhersagbaren Häufigkeit der Beitritts- und Aktualisierungsnachrichten, die von der vorherrschenden Netzwerkdynamik bestimmt wird. Sie berechnet sich aus der in einem Zeitraum zwischen t_0 und t_1 anfallenden Nachrichtenanzahl m und der Nachrichtengröße $N_{dynamic}$, die für Beitritts- und Aktualisierungsnachrichten gleich groß ist. Die Netzwerklast $L_{dynamic}$ im angegebenen Zeitraum zwischen t_0 und t_1, ausgedrückt durch $L_{dynamic}(t_0, t_1)$, berechnet sich aus

$$L_{dynamic}(t_0, t_1) = N_{dynamic} * m.$$

Die Nachrichtengröße $N_{dynamic}$ setzt sich dabei aus der Headerlänge l_{header}, einer IP-Adressenlänge $l_{address}$ und einer Dienstfeldlänge $l_{services}$ wie folgt zusammen

$$N_{dynamic} = l_{header} + l_{address} + l_{services}.$$

Aktualisierung, Konsistenz und Replikation QuickLink realisiert eine sehr hohe Replikationsrate der verwendeten Daten. Da jeder im Netzwerk befindliche QuickLink-Knoten über die gleiche Zyklusnachricht erreicht wird und die darin enthaltenen Daten in der QuickLink-List "Current" cacht, sind jederzeit die gleichen Daten auf allen Knoten vorhanden. Die eigene Datenbasis fördert die Autonomie der einzelnen Knoten, ohne dass das Netzwerk aufgrund der verwendeten Broadcastkommunikation stark belastet wird.

Bei der Verwaltung des Netzwerkes wird jedoch von einer kurzen Gültigkeit der Daten ausgegangen, die dem Zyklustakt, also der Sendefrequenz der Verwaltungsdaten, entspricht. Dabei wird die Strategie angewandt, dass eine Nachricht, die nicht empfangen wurde, auch nicht wiederholt gesendet wird. Dies ist sinnvoll, weil zum einen die zeitliche Gültigkeit der Nachricht und der darin enthaltenen Informationen nicht mehr gegeben ist und zum anderen mit hoher Wahrscheinlichkeit alle QuickLink-Knoten eine ausbleibende Broadcastnachricht nicht empfangen haben. Es ist daher besser, auf die nächste Nachricht mit neuen, aktuellen Daten zu warten. Damit wird auch die Konsistenz der Daten aller QuickLink-Knoten im Netzwerk sichergestellt. Mit der nächsten eintreffenden Nachricht aktualisieren alle QuickLink-Knoten gleichzeitig ihre Datenbasis.

Synchronisierung Die Synchronisation aller QuickLink-Knoten spielt eine große Rolle für das Zusammenwirken im QuickLink-Netzwerk. Alle Knoten müssen zwingend synchron laufen, damit immer zur richtigen Zeit Nachrichten gesendet und auch erwartet und Timing-Fehler vermieden werden.

Eine zentrale Synchronisation auf eine absolute Bezugszeit wäre sehr aufwändig und passt nicht gut in das Konzept der hochdynamischen QuickLink-Knoten. Daher erfolgt die Synchronisierung in QuickLink auf Basis der empfangenen Broadcastnachrichten. Jede empfangene Nachricht stellt neben einer Informationsquelle auch einen Synchronisationspunkt dar. Zu diesem Zeitpunkt werden alle Timer gestoppt und neu gestartet. Dadurch kann immer eine Synchronität aller QuickLink-Knoten sichergestellt werden. Ein neu ins Netzwerk eintretender QuickLink-Knoten muss daher auch erst mindestens eine Zyklusnachricht abwarten, bis er synchron läuft und selbst eine Zyklusnachricht absetzen darf. Da die Laufzeitunterschiede der Broadcastnachrichten im lokalen logischen Netzwerk im Verhältnis zu den verwendeten Karenzzeiten[12] - im Falle eines Zyklustaktes von

[12] Vgl. dazu auch Abschnitt 7.3.1.1, Softwarekomponente *Timer*, Seite 114.

einer Sekunde würde sie bei etwa 200 ms liegen - klein sind[13], spielen sie für die Synchronisation des QuickLink-Netzwerkes eine vernachlässigbare Rolle.

7.3.2 Funktionsweise und Vorgänge

Die Funktionsweise des CycleManager wird anhand der verschiedenen Vorgänge erklärt, die er zur Verwaltung des logischen QuickLink-Netzwerkes leistet. Sie werden in den nächsten Unterabschnitten näher beschrieben. Zusätzlich illustriert das Zustandsdiagramm in Abbildung 7.10, auf die sich in den Unterabschnitten bezogen wird, die funktionalen Zusammenhänge des CycleManagers.

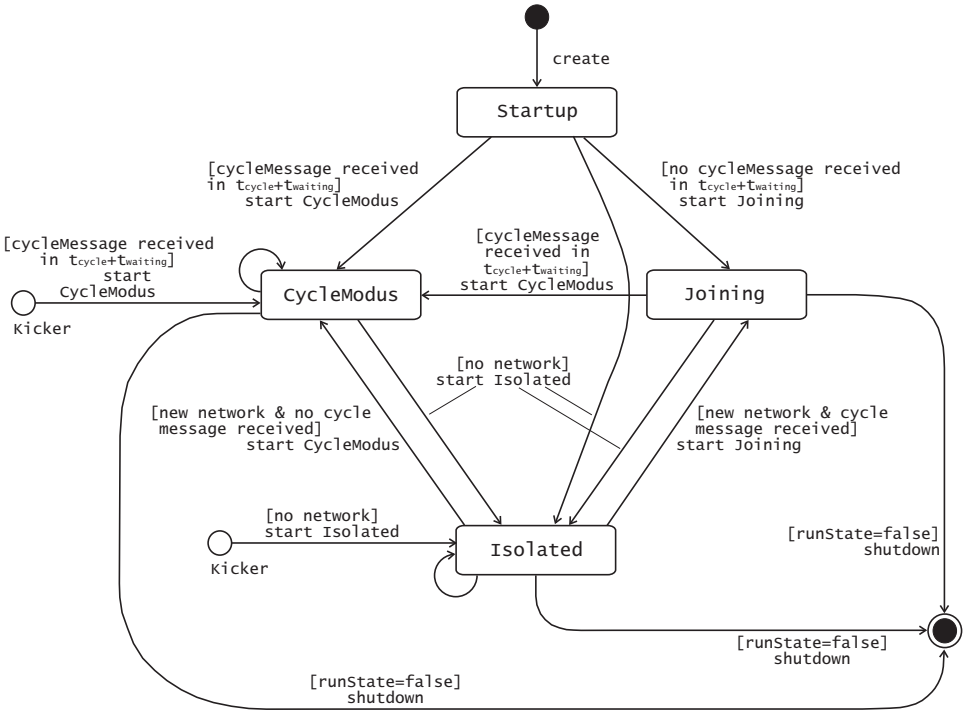

Abbildung 7.10: Das Zustandsdiagramm des CycleManagers

7.3.2.1 Bootstrapping und Netzwerkeintritt

Unter dem Begriff *Bootstrapping* versteht man im Umfeld der Informatik jeden Prozess, der aus einem einfachen System ein komplizierteres System akti-

[13] Nach eigenen Messungen [EDR04] liegen die Roundtripzeiten, gemessen mit dem Ping-Dienst, in lokalen Ethernet-Netzwerken bei einer Millisekunde und in einem Wireless LAN 802.11b im Infrastrukturmodus zwischen 3 und 4 Millisekunden. Die Laufzeit, d.h. die Ausbreitungszeit einer Nachricht im Netzwerk, entspricht der Hälfte der Roundtripzeit, wenn man symmetrischen Verbindungseigenschaften, also gleiche zeitliche Beschränkungen für den Hin- und Rückweg, unterstellt.

viert [Wik07a]. In der Welt der höheren logischen Netzwerke wird der Begriff für alle Vorgänge verwendet, die zur Inbetriebnahme eines Netzwerkes benötigt werden. Auch die Inbetriebnahme von QuickLink beginnt mit einer Bootstrapping-Phase. Nach der Erzeugung des CycleManager-Threads geht dieser in den Zustand *Startup* über. Dort versucht er seinerseits, alle von ihm benötigten und abhängigen Komponenten, wie in Abbildung 7.3 dargestellt, zu initialisieren. Können die UDP-Sockets nicht an einer gültigen IP-Adresse initialisiert werden, ist kein Netzwerk vorhanden und der CycleManager geht direkt, wie in Abbildung 7.10 zu sehen, in den Zustand *Isolated* über. Ist die Initialisierung hingegen erfolgreich verlaufen, lauscht er über die Receiver auf Netzwerknachrichten.

Nach diesen Vorgängen schließt sich eine Synchronisationsphase an, in der der CycleManager versucht, einem bestehenden Netzwerk beizutreten und sich mit ihm zu synchronisieren. Dazu startet der CycleManager seine timerNormal und wartet die Zeit $t_{normal} + t_{waiting}$ ab, um eine reguläre Zyklusnachricht eines anderen QuickLink-Knotens zu empfangen. Es gibt zwei Möglichkeiten, wie nun weiter verfahren werden kann.

Gelingt der Empfang einer Zyklusnachricht, besteht schon ein QuickLink-Netzwerk. Der CycleManager setzt mit Empfang der Zyklusnachricht zwecks Synchronisation seinen timerNormal zurück und startet diesen neu. Er ist jetzt im Takt mit dem bestehenden QuickLink-Netzwerk. Allerdings weiß dieses noch nichts von seiner Anwesenheit. Daher sendet er eine Beitrittsnachricht über den *updateSocket*, mit der er seine Anwesenheit bekannt macht und alle anwesenden QuickLink-Knoten ihre Datenbasis aktualisieren. Nach dem Senden und Empfangen seiner eigenen Beitrittsnachricht bricht der CycleManager den Bootstrapping-Prozess ab und geht in seinen normalen Zustand *CycleModus* über, in dem er die weitere Abwicklung seiner Einbindung über die Standardvorgänge vollziehen kann.

Empfängt der CycleManager keine Zyklusnachricht in der Zeit $2(t_{normal} + t_{waiting})$, geht er davon aus, dass kein anderes QuickLink-Netzwerk vorhanden ist und versucht, sein eigenes QuickLink-Netzwerk zu gründen. Dazu sendet er eine Beitrittsnachricht und empfängt diese selbst. Er fügt sich dann selbst in seine QuickLink-List als einzigen Knoten ein, sendet daraufhin eine Zyklusnachricht und geht in den Zustand *CycleModus* über.

Nach der Bootstrapping-Phase befindet sich der CycleManager also entweder im Zustand *CycleModus* oder im Zustand *Isolated*.

7.3.2.2 Verwaltung im Zustand CycleModus

Der Zustand *CycleModus* stellt den Standardzustand des CycleManagers dar. In ihm wird die Standardsituation, die Teilnahme an einem und Verwaltung eines QuickLink-Netzwerkes realisiert. Die ausgelösten Managementprozesse und Abläufe in diesem Zustand lassen sich in zwei große Situationen einteilen, deren

Eintreten durch den rechtzeitigen Empfang von Zyklusnachrichten entschieden wird. Als Voraussetzung wird angenommen, dass der CycleManager, und damit der ganze QuickLink-Knoten, mit dem QuickLink-Netzwerk synchron läuft. Dies muss vor den Übergängen in diesen Zustand sichergestellt werden.

Wird eine Zyklusnachricht in der Zeit $t_{normal} + t_{waiting}$ empfangen, so wird der Nachrichteninhalt in die QuickLink-List "Current" gespeichert, der timerNormal rückgesetzt und neu gestartet, um weiterhin synchron zu laufen. Als nächstes laufen zwei Vorgänge zur Pflege der Datenstrukturen simultan ab: Zum einen wird die Priority-List mit den Prioritätswerten des sendenden QuickLink-Knotens aktualisiert und zum anderen die QuickLink-Lists gepflegt. Dazu werden die Inhalte der beiden QuickLink-Lists "Old" und "Current" miteinander verglichen, um Änderungen im QuickLink-Netzwerk detektieren zu können. Die QuickLink-List "Old" repräsentiert dabei den Netzwerkzustand zum Zeitpunkt des vorherigen Zyklustaktes. Erkennbare Änderungen zum vorherigen Zustand können bezüglich

- der einzelnen QuickLink-Knoten selbst,
- der Anzahl der QuickLink-Knoten im Netzwerk,
- der von den Knoten angebotenen Dienste und
- der Leistungsparameter des sendenden Knotens

auftreten. Wenn Änderungen auftreten, wird die QuickLink-List "Old" durch die QuickLink-List "Current" ersetzt. Gleichzeitig werden alle Änderungen über verlorene oder hinzugekommene QuickLink-Knoten sowie über bestehende Knoten mit geänderten Diensten an QuickLink weitergeleitet, um andere Infrastrukturdienste und MAS durch Push-Nachrichten zu verständigen. Dafür ist ein eigenes Interface "PushedQuickLinkInfos"[14] vorgesehen. Abschließend wird aus der letzten Senderadresse und der aktuellen QuickLink-List der nächste Knoten (*Successor*) berechnet, der mit Senden an der Reihe ist und von dem die nächste Zyklusnachricht erwartet wird. Ist der Knoten selbst der *Successor*, sendet er selbst seine Zyklusnachricht nach Ablauf der Zeit t_{normal}.

Wird keine Zyklusnachricht in der Zeit $t_{normal} + t_{waiting}$ empfangen, tritt die andere Situation ein: Der CycleManager geht bei ausbleibender Zyklusnachricht davon aus, dass der im vorherigen Zyklus berechnete *Successor* aus dem Netzwerk ausgetreten ist. Daraufhin wird zuerst der timerShortened gestartet, der die zusätzlich zur t_{normal} abgelaufene Zeit $t_{waiting}$ berücksichtigt. Damit wird die Synchronität zum ursprünglichen Zyklustakt beibehalten. Nun wird zuerst überprüft, ob man selbst der Nachfolger des verloren gegangenen Knotens ist und damit als nächstes mit dem Senden einer Zyklusnachricht dran ist. Danach wird

[14] Die Interfaces QuickLinks werden im Abschnitt 7.5 ab Seite 132 behandelt.

der verloren gegangene Knoten aus der QuickLink-List "Old" und der Priority-List entfernt und die entsprechende Nachricht an QuickLink ausgelöst. Ist der Knoten selbst der *Successor*, sendet er nach Ablauf der Zeit $t_{shortened}$ selbst eine Zyklusnachricht. Zu dieser Zeit wird von allen QuickLink-Knoten im Netzwerk die nächste Zyklusnachricht erwartet.

Ein- und Austritte anderer QuickLink-Knoten sowie die Aktualisierungsnachrichten werden simultan zur Verarbeitung von Zyklusnachrichten im Zustand *CycleModus* abgehandelt. Sie sind in der Abarbeitung wesentlich weniger aufwändig als die Behandlung von Zyklusnachrichten, da die Datenumfänge solcher Nachrichten und die möglichen sich daraus ergebenden Konsequenzen geringer sind. In der Implementierung des CycleManagers ist sichergestellt, dass das Editieren und Lesen von QuickLink-Lists mit exklusiven Rechten erfolgt, damit sich die Vorgänge zur Behandlung der verschiedenen empfangenen Nachrichtenarten nicht gegenseitig behindern oder zu Inkonsistenzen führen.

Beim Empfang von Beitritts- oder Aktualisierungsnachrichten wird erst einmal anhand der Senderadresse und der QuickLink-List "Old" geprüft, ob sich der Knoten schon im QuickLink-Netzwerk befindet. Ist dies der Fall, war es eine Aktualisierungsnachricht und der dienstbezogene Teil des betreffenden Datensatzes in der QuickLink-List "Old" wird durch den dienstbezogenen Teil der Nachricht ersetzt. Ebenso wird die Priority-List aktualisiert. Ist der sendende Knoten kein Mitglied in der QuickLink-List "Old", so muss es sich um eine Beitrittsnachricht handeln. Der Knoten wird anhand seiner IP-Adresse an den richtigen Stellen der QuickLink-List "Old" und der Priority-List einsortiert und die Berechnung eines neuen *Successors* ausgelöst. Da diese Vorgänge auf allen QuickLink-Knoten gleichzeitig passieren, sind alle Knoten und ebenfalls die nächste gesendete Zyklusnachricht wieder up-to-date.

7.3.2.3 Eintritt ins Netzwerk aus dem Zustand Isolated

Der Zustand *Isolated* des CycleManagers in Abbildung 7.10 beschreibt die Situation, in der ein mobiles Endgerät keine Verbindung zu einem Netzwerk und daher ein QuickLink-Knoten auch keine Verbindung zu einem QuickLink-Netzwerk hat. In diesem Fall kann ein Knoten noch eine IP-Adresse haben, die aber nicht mehr gültig ist. Im Zustand *Isolated* betrachtet sich der QuickLink-Knoten wirklich als isoliert und verhält sich insoweit passiv, als dass er nur noch auf Nachrichten wartet und keine selbst aussendet. Er überprüft aber in regelmäßigen Abständen die Java VM auf eine neue IP-Adresse.

Empfängt der CycleManager wieder eine Zyklusnachricht, so muss er sich schon in einem anderen Netzwerk befinden und eine neue, gültige IP-Adresse zugewiesen bekommen haben. In diesem Fall geht er in den Zustand *Joining* über. Er versucht nun durch das Senden von Beitrittsnachrichten, sich im neuen QuickLink-Netzwerk bekannt zu machen. Gelingt dies, so kann er dies anhand

der Daten der nächsten Zyklusnachricht verifizieren und im positiven Fall in den Zustand *CycleModus* übergehen.

Hat der CycleManager über seine Java VM eine neue IP-Adresse detektiert, empfängt aber keine neuen Nachrichten, so geht er davon aus, dass er allein im Netzwerk ist. Er sendet darauf hin eine Zyklusnachricht, synchronisiert sich mit sich selbst und geht ebenfalls in den Zustand *CycleModus* über.

7.3.2.4 Austritt aus dem Netzwerk

Ein regulärer Austritt aus dem QuickLink-Netzwerk entspricht nicht der Philosophie des QuickLink-Netzwerkes, welches auf eine hohe Dynamik ausgelegt ist. Da es keine explizite Registrierung von QuickLink-Knoten im Netzwerk gibt, ist auch keine Deregistrierung vorgesehen. Solche Vorgänge werden im Infrastrukturdienst-Framework QuickLinkNet nur auf der Ebene der Anwendungsdienste, die nicht in den Bereich des Infrastrukturdienstes QuickLink fallen, ausgeführt.

Das einzige Szenario für einen expliziten Austritt aus dem QuickLink-Netzwerk wäre das Beenden des Infrastrukturdienstes QuickLink auf einem Knoten seitens des Benutzers. In diesem Fall kann der QuickLink-Knoten eine Aktualisierungsnachricht absetzen, bei der hilfsweise das Feld 3 des Headers, der die prozentuale Auslastung des Speichers der Java VM angibt und immer positiv ist, auf minus eins gesetzt wird und damit den bevorstehenden Austritt signalisiert. Der Knoten wird daraufhin von allen QuickLink-Knoten aus ihren Datenbasen gelöscht und sicherheitshalber der *Successor* neu berechnet. Die Zeit t_{gone} bis zum Detektieren eines solchen geplanten Austritts beträgt dann genau

$$t_{gone} = t_{normal}.$$

7.3.2.5 Plötzliches Verschwinden

Das plötzliche Verschwinden aus dem QuickLink-Netzwerk ist viel wahrscheinlicher für hochdynamische Knoten als der vorhersehbare Austritt. Dieser Vorgang wird aber vom verbleibenden QuickLink-Netzwerk nicht direkt bemerkt, da der unfreiwillig ausgetretene QuickLink-Knoten in dieser Situation keine Nachricht mehr absetzen kann. Der plötzliche Austritt aus dem QuickLink-Netzwerk wird daher durch die Abläufe im CycleModus erst bemerkt, wenn der verschwundene Knoten mit dem Senden der Zyklusnachricht an der Reihe ist. Dies kann im schlechtesten Fall erst nach dem Durchlauf einer ganzen QuickLink-List passieren. Die Zeit t_{lost} bis zum Detektieren des Austritts beträgt dann mit n Knoten im Augenblick der Detektion

$$t_{lost} = n * t_{normal} \ ,$$

bei einer t_{normal} von einer Sekunde, also n Sekunden.

7.4 Die Softwarekomponente Kicker

7.4.1 Self-Healing

Begriff Der Begriff *Self-Healing* (Selbstheilung) bezeichnet die Eigenschaft eines Systems, sich selbst auch ohne fremdes Eingreifen in einem verfügbaren Zustand zu halten [SPTU07a]. Im Gegensatz zum Failover [Wik07c], bei dem die Hochverfügbarkeit im Mittelpunkt steht und mit erheblichem Aufwand, i.d.R. durch eine doppelte Auslegung [Wik07d] von Netzwerken, Servern und Hardware, realisiert wird, versucht man mit Self-Healing-Techniken, fehlerhafte Zustände des Systems durch das System selbst erkennen und beheben zu lassen.

In verteilten Systemen spielt die Fehlerentdeckung und ihre Fehlerbeseitigung [SPTU07b] als Heilung im Kontext des Self-Healing eine große Rolle. Gerade bei hochdynamischen Systemen, wie QuickLinkNet es darstellt, ist die Wahrscheinlichkeit des Auftretens unvorhergesehener Zustände recht groß. Daher ist der Einsatz von Self-Healing Techniken hier sehr sinnvoll.

Einsatzszenarien für Self-Healing QuickLink ist, wie in Abschnitt 7.1.4 beschrieben, in mehreren Threads pro QuickLink-Knoten implementiert, welche lokal untereinander kommunizieren. Einzelne Threads sind aber auch in die Netzwerkkommunikation eingebunden, empfangen also die Nachrichten anderer QuickLink-Knoten.

An dieser Stelle ergeben sich potentielle Fehlermöglichkeiten: Zum einen kann die UDP-Kommunikation gestört werden, wodurch der Inhalt der Nachrichten verfälscht oder nicht vollständig übertragen werden kann. Zum anderen können bei der Broadcastkommunikation trotz der robusten Synchronisationsmethode[15] Timing-Fehler entstehen. Um die Auswirkungen dieser potentiellen Fehler zu beheben, wurde der Kicker als zusätzliche Softwarekomponente in QuickLink eingeführt.

7.4.2 Die konkrete Aufgabe Kickers

Treten im QuickLink-Netzwerk gravierende Timing-Fehler oder verstümmelte Nachrichten auf, kann es in ungünstigen Konstellationen vorkommen, dass sich der Haupt-Thread des CycleManagers eines QuickLink-Knotens verklemmt. Der QuickLink-Knoten kann dann keine Informationen aus dem QuickLink-Netzwerk mehr verarbeiten und nicht mehr an der Netzwerkkommunikation teilnehmen, obwohl seine Receiver noch Daten empfangen. Diese Situation tritt allerdings äußerst selten auf. Für die korrekte Funktionsweise QuickLinkNets ist es aber wichtig, dass ein solcher QuickLink-Knoten sich wieder an der Netzwerkkommu-

[15] Vgl. dazu auch Abschnitt 7.3.1.5.

nikation beteiligt. Dies sollte ohne Eingreifen des Nutzers und ohne Neustart des QuickLink-Dienstes funktionieren.

Um zu gewährleisten, dass solche Fehler automatisch erkannt und behoben werden, wurde der Kicker implementiert.

7.4.3 Funktionsweise

Der Kicker läuft als eigener Java-Thread innerhalb QuickLinks und kommuniziert mit dem CycleManager. Er überwacht dessen Zustand auf Abnormitäten und reagiert, wenn eine solche auftritt. Abbildung 7.11 zeigt das Zustandsdiagramm von Kicker.

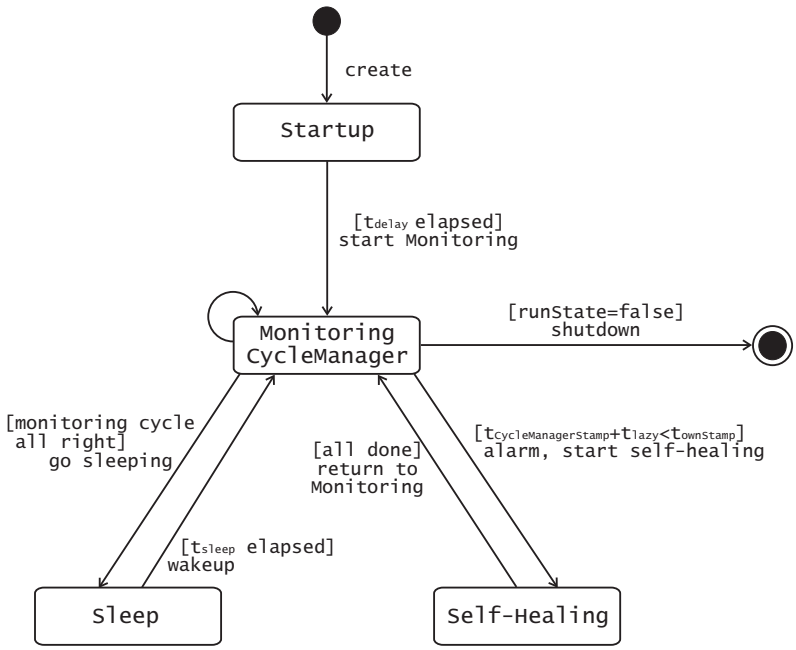

Abbildung 7.11: Das Zustandsdiagramm der Softwarekomponente Kicker

Der Zustand *Startup* beinhaltet nach der Initialisierung Kickers eine gewisse Verzögerungszeit, bis Kicker wirklich losläuft, um dem CycleManager Gelegenheit zu geben, in einen stabilen Zustand zu gelangen. Die Verzögerungszeit t_{delay} bemisst sich dabei nach der vom CycleManager verwendeten Zykluszeit t_{cycle} und beträgt das Fünfzehnfache dieser. Nach Ablauf dieser Phase geht der Kicker in den Zustand *Monitoring CycleManager* über, in der er seine Überwachungsfunktion ausübt. Sind alle überwachten Parameter in Ordnung, legt sich Kicker für die Zeit t_{sleep} schlafen, indem er in den Zustand *Sleep* übergeht. Ist die Schlafzeit abgelaufen, geht Kicker erneut in den Zustand *Monitoring CycleManager* über und überprüft den CycleManager erneut. Die Schlafzeit ist ebenfalls von

der Zykluszeit des CycleManagers abhängig und wurde auf

$$t_{sleep} = \frac{3}{5}\, t_{cycle}$$

festgelegt. Mit dieser Wahl von t_{sleep} werden zwei Dinge sichergestellt: Zum einen überprüft Kicker den CycleManager Thread mindestens einmal pro Zyklusdurchlauf, zum anderen werden aufeinanderfolgende Überprüfungen nie in den gleichen zeitlichen Ablaufpunkten des CycleManager Threads durchgeführt. Eine Synchronisation zwischen den Überprüfungszeitpunkten Kickers und den inneren Ablaufsituationen des CycleManagers kann somit vermieden werden, was die Robustheit der Überwachungsfunktion zusätzlich erhöht.

Zum Erkennen von schwerwiegenden Fehlern wird im Haupt-Thread des CycleManagers bei jedem Durchlauf die aktuelle Systemzeit als Zeitstempel (*Timestamp*) $t_{CycleManagerStamp}$ festgehalten, der beim normalen Ablauf des CycleManagers mehrmals pro Sekunde aktualisiert wird. Kicker vergleicht im Zustand *Monitoring CycleManager* $t_{CycleManagerStamp}$ mit seiner eigenen Systemzeit $t_{ownStamp}$ unter Berücksichtigung einer Verzögerungszeit t_{lazy} mit $t_{lazy} = 3\, t_{cycle}$. Wird die Zeitschranke

$$t_{CycleManagerStamp} + t_{lazy} < t_{ownStamp}$$

überschritten, löst Kicker sein Notfallprogramm aus und geht in den Zustand *Self-Healing* über, indem der Selbstheilungsmodus ausgelöst wird. Im Selbstheilungsmodus führt Kicker alle Maßnahmen aus, um den CycleManager wieder in einen definierten Zustand zu versetzen und am QuickLink-Netzwerk teilnehmen zu lassen. Dazu werden die Aufgaben in folgender Reihenfolge erfüllt:

1. Unterbrechung und Rücksetzen der Timer,
2. Unterbrechung und Rücksetzen der Receiver,
3. Unterbrechung und Rücksetzen des CycleManagers,
4. Rücksetzen aller nötigen Systemzustände,
5. Starten der Receiver,
6. Synchronisierung nach Empfang einer Zyklusnachricht und
7. Neustart der Timer und Überführen des CycleManagers in den Zustand *CycleModus*.

Nach diesem Ablauf sollte der CycleManager wieder synchron mit dem QuickLink-Netzwerk laufen und regulär an ihm teilnehmen. Ist der betreffende QuickLink-Knoten mittlerweile aus dem QuickLink-Netzwerk ausgeschieden, so kann er dies

durch die Auswertung der empfangenen Zyklusnachricht schnell erfassen und durch eine Beitrittsnachricht schnell korrigieren. Falls der CycleReceiver nach drei Zyklen überhaupt keine Zyklusnachricht empfangen hat, startet Kicker den CycleManager im Zustand *Isolated*, von wo aus der QuickLink-Knoten wieder die normalen Maßnahmen zum Detektieren eines Netzwerkes und Beitreten zu einem solchen aufnimmt.

7.5 Die Interfaces

Über die Interfaces können andere Softwaremodule QuickLinkNets, Dienste und MAS auf die Funktionalitäten QuickLinks zugreifen. Dabei fungiert QuickLink in erster Linie als Informationslieferant. Die verschiedenen Informationsabnehmer benötigen unterschiedlich detaillierte Informationen. QuickLink stellt daher fünf Interfaces zur Verfügung, die Informationen und Funktionalität in unterschiedlicher Tiefe anbieten. Diese sind die Interfaces

- Priorities,
- Performance,
- Quicklink,
- OwnServices und
- PushedQuickLinkInfos.

7.5.1 Priorities

Für den internen Gebrauch QuickLinks und auf technischer Ebene reichen die Prioritätswerte der Leistungsparameter im Byte-Format. Diese werden aus den gemessenen Leistungsparametern des eigenen QuickLink-Knotens errechnet und stellen eine inhaltlich reduzierte, aber ausreichende Information dar. Sie werden über das Interface *Priorities* angeboten. Dieses bietet folgende Leistungsparameter im Datentyp *Byte* an:

- aktuell verfügbare Rechenleistung der Java VM (als Vergleichswert zu anderen QuickLink-Knoten),
- aktuell freier Speicher der Java VM in MByte,
- maximal verfügbarer Speicher der Java VM in jeweils 10 MByte,
- aktuelle Speicherauslastung der Java VM in Prozent bezogen auf den maximal verfügbaren Speicher und

- die Aufenthaltszeit des QuickLink-Knotens im lokalen QuickLink-Netzwerk in Minuten.

Da Prioritätswerte auch für die Steuerung und Verwaltung des QuickLink-Netzwerkes durch den Infrastrukturdienst *ServiceJuggler* verwendet werden, bietet das Interface Priorities noch zwei weitere Methoden an, welche speziell für die Wahl des Regions-Managers (über die Auswertung eines Prioritätswerts) verwendet werden[16]. Die beiden Methoden beziehen ihre Rückgabewerte direkt aus der *Priority-List* des CycleManagers, welche mit den Prioritätswerten aus den empfangenen Zyklusnachrichten gefüllt und aktualisiert wird. Sie liefern

- die IP-Adresse des Knotens mit dem höchsten (zur Regions-Managerwahl verwendeten) Prioritätswert im QuickLink-Netzwerk im *String*-Format und
- die zur Knotenwahl verwendete Priorität (als *Byte*) eines beliebigen Knotens im QuickLink-Netzwerk, dessen IP-Adresse zu diesem Zweck im *String*-Format als Eingabeparameter übergeben werden muss.

Die über das Interface *Priorities* abrufbaren Leistungsparameter im *Byte*-Format dienen in erster Linie internen Zwecken, können aber ebenso von MAS und andere Diensten, welche direkt mit diesen Werten arbeiten wollen, genutzt werden.

7.5.2 Performance

Das Interface *Performance* liefert deutlich detailliertere Informationen über die Leistungsparameter der Java VM des QuickLink hostenden Knotens und reizt die Möglichkeiten des PerformanceAnalysers[17] bzgl. der Auflösung der Leistungsparameter aus. Die Werte dienen Anwendungen, welche genaue Leistungswerte benötigen, wie z.B. systemfremden Diensten zur Netzwerküberwachung oder Lastverteilung. Das Interface *Performance* bietet folgende leistungsbezogene Werte im Datentyp *Long* an:

- aktuell verfügbare Rechenleistung der Java VM,
- maximal verfügbarer Speicher der Java VM in KByte,
- aktuell freier Speicher der Java VM in KByte,
- aktuell allozierter Speicher der Java VM in KByte,
- aktuelle Speicherauslastung der Java VM in Prozent (als Datentyp *Float*) bezogen auf den maximal verfügbaren Speicher,

[16] Der Infrastrukturdienst ServiceJuggler und die Wahl des Regions-Managers wird in Kapitel 8 behandelt.

[17] Der PerformanceAnalyser wird im nächsten Abschnitt 7.6 behandelt.

- die Größe des aktuell von QuickLink verwendeten Speichers in KByte und
- die Aufenthaltszeit des QuickLink-Knotens im lokalen QuickLink-Netzwerk in Sekunden.

7.5.3 Quicklink

Das Interface *Quicklink* ist speziell auf die Kommunikation zwischen QuickLink und ServiceJuggler zugeschnitten, kann aber auch von anderen Diensten genutzt werden. Es bietet z.B. auch mobilen Agentensystemen die Möglichkeit, direkt auf die Dienstinformationsbasis der QuickLink-List "Old" zuzugreifen und die Adressen anderer QuickLink-Knoten entsprechend ihrer laufenden Dienste abzufragen. Die Informationen sind dabei immer aktuell, d.h. nicht älter als t_{cycle}. Dazu bietet das Interface Quicklink folgende Abfragemöglichkeiten:

- Die IP-Adresse im Datentyp *String* eines QuickLink-Knotens im QuickLink-Netzwerk, der den als String übergebenen Parameter "Dienst" hostet und aktuell ausführt,
- die IP-Adressen aller QuickLink-Knoten im QuickLink-Netzwerk als Datentyp *String-Array*, die den als String übergebenen Parameter "Dienst" hosten und aktuell ausführen und
- eine Methode, um QuickLink darüber zu informieren, dass die ServiceJuggler-Instanz des eigenen Knotens nun den ServiceRegistry ausführt und damit zum Regions-Manager geworden ist.

Speziell die letzte Methode ist ServiceJuggler-spezifisch und für alle verteilten Anwendungen, welche auf Basis von Anwendungsdienstinformationen arbeiten (wie MA und MAS), sehr wichtig. Sie wird dazu verwendet, den Verzeichnisdienst für Anwendungsdienste *ServiceRegistry* sofort bekannt zu machen und QuickLink zu einer Update-Nachricht zu veranlassen. Damit wird der Dienst und die neue Rolle des QuickLink-Knotens als Regions-Manager sofort und ohne Verzögerung verbreitet.

7.5.4 OwnServices

Das Interface OwnServices ist speziell für das MAS auf dem QuickLink-Knoten geschrieben worden, kann aber auch für die Anbindung anderer verteilter Anwendungen verwendet werden. Es hat zwei Aufgaben: Zum einen dient es zur Abfrage der eigenen, in QuickLink registrierten Infrastrukturdienste und grundlegenden Dienste, zum anderen bietet es die Möglichkeit, diese Dienste im QuickLink-Netzwerk an- und abzumelden.

Dazu bietet OwnServices Methoden an, die Folgendes ermöglichen:

- Die Abfrage der Namen der möglichen Dienste, welche überhaupt registriert werden können, also im QuickLink-Netzwerk bekannt sind. Diese werden als *String-Array* zurückgeliefert.
- Die Abfrage der Namen der aktuell als "werden ausgeführt" registrierten Dienste des eigenen QuickLink-Knotens, welche ebenfalls als *String-Array* zurückgeliefert werden.
- Eine (Setz-)Methode zum An- und Abmelden eines Dienstes. Der Dienstname wird als *String* und der gewünschte Status als *Boolean* übergeben.
- Eine (Setz-)Methode zum An- und Abmelden mehrerer Dienste gleichzeitig. Die Dienstnamen werden als *String-Array* und die gewünschten Zustände als *Boolean-Array* übergeben.

Bei der Verwendung der Setzmethoden für Dienste wird jedesmal durch QuickLink und den CycleManager eine Aktualisierungsnachricht ausgelöst. Um eine Flut dieser Nachrichten bei der Veränderung mehrerer Dienstzustände gleichzeitig zu vermeiden, wurde die letztere Setzmethode implementiert, die nur eine Aktualisierungsnachricht für alle veränderten Dienste auslöst.

Mit den Möglichkeiten des Interfaces OwnServices können die in QuickLink bekannten und verwalteten Dienstinformationen vollständig und effizient administriert werden.

7.5.5 PushedQuickLinkInfos

Die Verwendung des Interfaces PushedQuickLinkInfos ist optional. Es ist das einzige Interface, welches von anderen Komponenten implementiert werden muss. Dies ist notwendig, um am Push-Dienst von QuickLink teilnehmen zu können. Der Push-Dienst stellt sicher, dass registrierte Komponenten sofort von Änderungen, die im QuickLink-Netzwerk detektiert werden, benachrichtigt werden und die Daten der Änderungen übertragen bekommen. Der Push-Dienst liefert folgende Daten:

- Die IP-Adresse eines verlorenen QuickLink-Knotens als *String*.
- Die IP-Adresse eines neuen QuickLink-Knotens als *String* und seine laufenden Dienste als *String-Array*.
- Bei Änderungen die IP-Adresse des QuickLink-Knotens als *String*, die geänderten Dienste als *String-Array* und den Status seiner geänderten Dienste als *Boolean-Array*.

Durch die Verwendung eines ereignisgetriebenen Push-Dienstes kann die Aktualisierungsgeschwindigkeit QuickLinks auch auf andere Dienste und das MAS übertragen werden, was die Leistungsfähigkeit des gesamten Systems bzw. der gesamten verteilten Anwendung verbessert. So kann bei der Verwendung dieses Push-Dienstes die Aktualisierung der Datenbasis der Routingdienste verbessert und damit die Gesamtperformance mobiler Agenten erhöht werden.

7.6 Die Softwarekomponente PerformanceAnalyser

Nachdem nun alle netzwerkbezogenen Komponenten QuickLinks behandelt wurden, wird mit dem *PerformanceAnalyser* eine Softwarekomponente vorgestellt, welche die Bestimmung der Leistungsfähigkeit der Plattform eines QuickLink-Knotens zur Aufgabe hat. Eine Aussage über die Leistungsfähigkeit der unterliegenden Plattform wird zwar auf der logischen Ebene des Infrastrukturdienstes QuickLink nicht benötigt, wohl aber auf der nächsten logischen Ebene, die durch den im nächsten Kapitel behandelten Infrastrukturdienst *ServiceJuggler* realisiert wird.

7.6.1 Aufbau und Architektur

Der PerformanceAnalyser übernimmt die Aufgabe der Leistungsmessung im Infrastrukturdienst QuickLink. Abbildung 7.12 zeigt den Aufbau des PerformanceAnalysers und seiner Komponenten. Zur Leistungsbestimmung liest der PerformanceAnalyser die Werte, die sich aus der Java VM heraus bestimmen lassen, direkt aus. Dies sind im Einzelnen die Werte:

- maximal verfügbarer Speicher der Java VM,
- aktuell freier Speicher der Java VM,
- aktuell allozierter Speicher der Java VM,
- aktuelle Speicherauslastung der Java VM in Prozent (als Datentyp *Float*) bezogen auf den maximal verfügbaren Speicher und
- die aktuelle Systemzeit als Bezugspunkt für die Berechnung der Aufenthaltszeit des QuickLink-Knotens im lokalen QuickLink-Netzwerk.

Da sich die Rechenleistung so nicht bestimmen lässt, kommt hierfür ein eigenes Leistungsmessmodul namens *Benchmark* zum Einsatz. Dabei ist die Komponente nicht als Benchmark im herkömmlichen Sinne zu verstehen, sondern als ein an die Einsatzbedingungen[18] QuickLinks angepasstes *Leistungsmessmodul*.

[18] Vgl. hierzu Abschnitt 7.1.3 auf Seite 109.

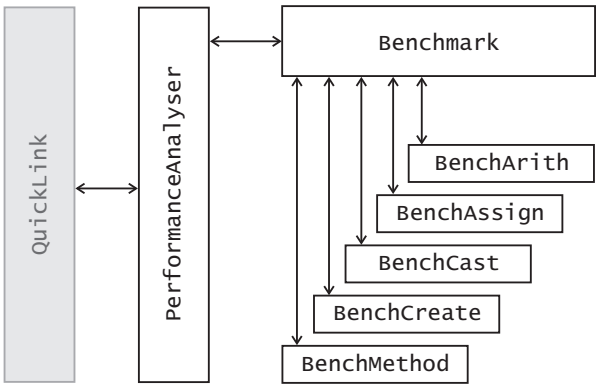

Abbildung 7.12: Die funktionale Struktur des PerformanceAnalysers

Der PerformanceAnalyser selbst ist als Java-Thread implementiert, der regelmäßig von QuickLink zum Leben erweckt wird, um die Performance-Messungen durchzuführen. Er startet wiederum den Thread Benchmark zur Leistungsmessung. Der Benchmark führt auf Anforderung des PerformanceAnalysers die Leistungsmessung aus, indem er seinerseits entsprechende Messobjekte erstellt, deren Methoden ausführt, die zurückgelieferten Teilergebnisse zusammenfügt und als Leistungskennzahl an den PerformanceAnalyser zurückliefert. Die einzelnen *Analysekomponenten (Analyser)*[19]

- BenchArith,
- BenchAssign,
- BenchCast,
- BenchCreate und
- BenchMethod

erzeugen dabei verschiedene Arten von Rechnenlast für einen Prozessor. Da das Lastprofil[20] beliebiger verteilter Anwendungen und Dienste nicht vorhersehbar ist, wurden die Analyser so ausgewählt, dass sie die Art von Last erzeugen, wie wir sie von verteilten Anwendungen hauptsächlich erwarten: Objekte anlegen, Methodenaufrufe von Objekten, Typumwandlungen primitiver Datentypen, Variablenzuweisungen und arithmetische Operationen auf primitiven Datentypen ausführen.

[19] Die Analysekomponenten basieren auf der *Java Grande Forum Benchmark Suite* [EPC07b], *Section 1*, sind aber modifiziert worden, um sie den Bedingungen QuickLinks anzupassen. Nähere Angaben zu den verwendeten Testmethoden sind im Anhang A.2.4 zu finden.

[20] Gemeint ist hier die konkrete Zusammensetzung und Intensität der von einer verteilten Anwendung erzeugten Rechenlastarten.

7.6.2 Funktionsweise

Nachdem QuickLink alle seine Komponenten, unter ihnen PerformanceAnalyser, aufgerufen und initialisiert hat, sind diese einsatzbereit. Zur Steuerung des PerformanceAnalysers verwendet QuickLink eine bestimmte Zeitspanne, das Messintervall, das in der Konfigurationsdatei angegeben werden kann. Es soll als Richtwert zwischen etwa 10 und 60 Sekunden liegen, kann aber für jeden QuickLink-Knoten individuell eingestellt werden. QuickLink triggert nun regelmäßig entsprechend des Messintervalls den PerformanceAnalyser an, welcher dann eine Leistungsmessung auslöst. Zu diesem Zweck ruft der PerformanceAnalyser alle Werte ab, die er direkt aus der Java VM auslesen kann und weckt den Benchmark-Thread, der wiederum nacheinander die Analyser aufruft und ihre Rückgabewerte sammelt. Die Analyser erzeugen dabei folgende Leistungswerte:

- BenchArith - misst die Leistung des Knotens bzgl. arithmetischer Operationen (add, mult, div) auf primitive Datentypen (Int, Long,Float, Double). Performanceeinheiten sind Additionen, Multiplikationen und Divisionen pro Messzeit.
- BenchAssign - misst die Leistung des Knotens bzgl. Zuweisungen verschiedener Typen von Variablen wie Skalare, Arrays, lokale Variable, Instanzvariablen und Klassenvariablen. Performanceeinheit ist die Anzahl von Zuweisungen pro Messzeit.
- BenchCast - misst die Leistung des Knotens bzgl. verschiedener, primitiver Datentypumwandlungen in Umwandlungen pro Messzeit.
- BenchCreate - misst die Leistung des Knotens bzgl. verschiedenartiger Generierungen von Arrays und Objekten. Performanceeinheiten sind Arrays bzw. Objekte pro Messzeit.
- BenchMethod - misst die Kosten von Methodenaufrufen. Die Methoden sind Instanz-, finale Instanz- und Klassenmethoden, die von derselben Instanz und einer anderen aufgerufen werden. Performanceeinheit ist die Anzahl von Aufrufen pro Messzeit.

Sind die Messungen aller Analyser beendet, so berechnet Benchmark das Gesamtergebnis aus deren Einzelergebnissen. Die implementierte Methodik kann im Anhang A.2.4 nachgelesen werden. Das Gesamtergebnis wird als Performancewert an den PerformanceAnalyser zurückgegeben, welcher nun alle seine Leistungswerte an QuickLink zurückgibt. QuickLink wiederum berechnet aus den Leistungswerten die aktuellen Prioritätswerte des QuickLink-Knotens, welche

u.a. in den Zyklusnachrichten[21] des Knotens mitgeschickt werden. Als letztes wird der Eintrag des eigenen Knotens in der Priority-List des CycleManagers mit den neuen Prioritätswerten aktualisiert.

Nach vollendeter Leistungsmessung verweilen die beiden Java-Threads PerformanceAnalyser und Benchmark im Zustand WAITING, in welchem sie auf ihren erneuten Aufruf warten. Da sie während des Messintervalls nun nicht mehr aufgerufen werden, beanspruchen sie auch keine Rechenleistung in dieser Zeit.

7.7 Zusammenfassung

Der Infrastrukturdienst QuickLink bildet ein begrenztes, lokales, logisches Netzwerk zwischen QuickLink-Knoten. Er ist auf die Abbildung einer hohen Netzwerkdynamik ausgelegt. Die zwischen den QuickLink-Knoten ausgetauschten Informationen beinhalten die IP-Adressen als Kommunikationsendpunkte der Plattformen, eine Reihe explizit bekannter Infrastrukturdienste und grundsätzlicher Dienste von MAS sowie leistungsbezogene Prioritätswerte der Plattformen. Als Kommunikationsmittel werden ausschließlich UDP-Broadcast-Nachrichten eingesetzt, wodurch die Netzwerkbelastung zur Verwaltung des Netzwerkes sehr gering gehalten werden kann.

Durch QuickLink ist die grundsätzliche Vernetzung von Plattformen im lokalen Bereich realisiert worden. Im folgenden Kapitel wird sich nun ServiceJuggler zugewendet, einem Infrastrukturdienst, der auf Basis von QuickLink die intelligente dynamische Lastverteilung bestimmter höhere Dienste und damit eine Selbstadaption des lokalen logischen Netzwerkes an sich ändernde Umgebungsbedingungen ermöglicht.

[21] Die Prioritätswerte des QuickLink-Knotens werden im Header seiner Zyklusnachrichten mitgeschickt, vgl. auch Abschnitt 7.3.1.4.

8 Anwendungsdienstinformationen einer Region verwalten - ServiceJuggler

Dieses Kapitel befasst sich mit dem Infrastrukturdienst *ServiceJuggler* als zweite Komponente des komplexen Infrastrukturdienstes QuickLinkNet. Wie in Abbildung 8.1 dargestellt, ist ServiceJuggler zwischen den beiden anderen Infrastrukturdiensten *QuickLink*, der für die lokale Netzwerksicht verantwortlich ist, und APLICOOVER, der die globale Sicht auf das Netzwerk realisiert, angeordnet.

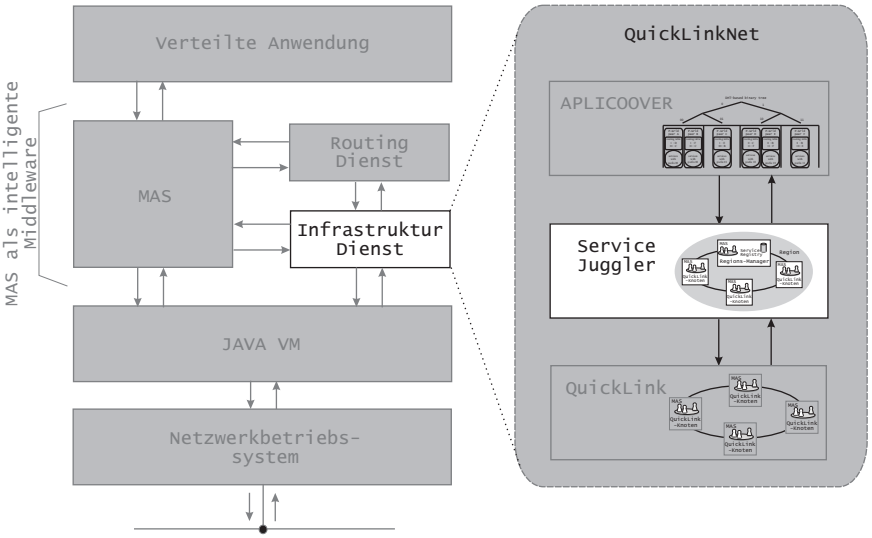

Abbildung 8.1: Der Infrastrukturdienst ServiceJuggler im Framework QuickLinkNet

Diese Anordnung korrespondiert mit der Funktion ServiceJugglers, die beiden Sichten auf das Netzwerk miteinander zu verbinden. Im Unterschied zur eingeschränkten Sichtweise QuickLinks, welches nur die Infrastrukturdienste und die *grundsätzlichen Dienste* eines MAS im lokalen logischen Netzwerk kennt, verwaltet ServiceJuggler *Anwendungsdienste* im lokalen Netzwerk, also Dienste, die mobile Agenten zur Abarbeitung ihres User-Tasks benötigen. Zusammengefasst realisiert ServiceJuggler in *QuickLinkNet* folgende Aufgaben:

- Die Anwendungsdienste in einer Region verwalten,
- die Region selbst verwalten und an dynamische Veränderungen adaptieren und

- die lokale und globale Sicht auf das gesamte logische Netzwerk intelligent miteinander verbinden.

Das Kapitel ist in sechs Abschnitte aufgeteilt. In Abschnitt 8.1 werden die einzelnen Aufgaben ServiceJugglers analysiert und die Randbedingungen, unter denen er arbeitet, diskutiert. Ferner widmet sich der Abschnitt dem architekturellen Aufbau ServiceJugglers, seiner Komponenten und der verwendeten Technologien und Standards. Die darauf folgenden drei Abschnitte 8.2 bis 8.4 behandeln die drei Module ServiceJugglers detaillierter. Abschnitt 8.5 behandelt die nach außen angebotenen Schnittstellen ServiceJugglers. Eine Zusammenfassung in Abschnitt 8.6 schließt dieses Kapitel ab.

8.1 Der Infrastrukturdienst ServiceJuggler im Überblick

8.1.1 Vorbetrachtung und Analyse zu Anwendungsdienste verwalten

8.1.1.1 Abstraktion der Dienstarten

Infrastrukturdienste und grundsätzliche Dienste eines MAS erfüllen Funktionalitäten, die eine rein technische Unterstützung[1] für den MA leisten, also für sein technisches Funktionieren notwendig sind. Anwendungsdienste hingegen erfüllen Funktionalitäten, die die im User-Task eines mobilen Agenten definierte Aufgabe direkt lösen helfen. Ein autonom handelnder, mobiler Agent braucht zur Lösung seiner Aufgabe beide Informationsarten:

- Informationen über technische Dienste seiner Umgebung, die seine Problemlösungsmöglichkeiten bestimmen und seine Funktionen erweitern, also bestimmen, WIE er eine Aufgabe umsetzen kann, und
- Informationen über Anwendungsdienste, also darüber, WAS für Aufgaben oder Teilaufgaben er lösen kann. Da mobile Agenten Teil der Middleware sind und entfernte Dienste selbst besuchen, brauchen sie noch Informationen, wo sich die gewünschten Dienste befinden, also WO eine Aufgabe umgesetzt werden kann.

Die Motivation für die getrennte Betrachtung der beiden Dienstarten, die auf unterschiedlichen Dynamikaspekten[2] und Abstraktionsniveaus[3] beruhen, wurde in Kapitel 5 ausführlich dargestellt. Anwendungsdienste unterliegen demnach einer geringeren Dynamik als technische Dienste und werden deshalb aus Performancegründen getrennt verwaltet. QuickLink ist, wie im Kapitel 7 beschrieben, für die

[1] Vgl. dazu auch Abschnitt 5.2.1.
[2] Vgl. hierzu auch Abschnitt 5.2 und seine Unterabschnitte.
[3] Vgl. dazu Abschnitt 6.3.

Verwaltung der *technischen Informationen* zuständig. ServiceJuggler ist dagegen für die Verwaltung von *Anwendungsdienstinformationen* im lokalen Netzwerkbereich verantwortlich. Durch die geringere Dynamik der Anwendungsdienste unterliegt eine Verwaltung der Anwendungsdienstinformationen auch geringeren Anforderungen an die Aktualität. Dafür ist eine zuverlässige An- und Abmeldung der Dienste, eventuell verbunden mit einer Authentifizierung, notwendig, um die Zuverlässigkeit und Qualität des Dienstangebotes sicherzustellen.

8.1.1.2 Lösungen für die Verwaltung von Anwendungsdienstinformationen

Die Verwaltung von Informationen in einem Netzwerk ist keine neue Aufgabe. Es existieren heute viele ausgereifte und implementierte Lösungen, die unter dem Begriff Verzeichnisdienst bekannt sind. In der Internetwelt existieren z.B. für lokale Netzwerkbereiche das *Active Directory* [Mic03] für Windows-Netzwerke, das *eDirectory* (ehemals *NDS*) [Nov02] für Novell-Netzwerke und der *Network Information Service (NIS)* [Sun02] für Unix-Netzwerke. Im globalen Internetbereich gibt es *Universal Description, Discovery and Integration (UDDI)* [Org04] und das *Domain Name System (DNS)* [Moc87b]. Bei der Umsetzung eines Verwaltungsdienstes für Anwendungsdienste, dessen funktionaler Großteil durch einen Verzeichnisdienst abgebildet werden kann, sollte daher auf bestehende Standards zurückgegriffen werden.

Interessant ist im lokalen Netzwerkbereich vor allem das standardisierte und in der Internetwelt weit verbreitete Kommunikationsprotokoll *Lightweight Directory Access Protocol (v3) (LDAP)* [WHK97], welches extra für die Kommunikation mit Verzeichnisdiensten entwicklelt wurde und auf dem heute fast alle modernen Lösungen von lokalen Verzeichnisdiensten im Internet beruhen. Es sollte daher in einer Anwendungsdienstverwaltung als Austauschprotokoll zum Einsatz kommen, um eine größtmögliche Flexibilität zu erreichen. Bestehende Implementierungen von LDAP-Verzeichnisdiensten gehen allerdings von statischen Netzwerkbedingungen aus und sind nicht auf das Schreiben, sondern auf das reine Lesen von Verzeichniseinträgen [CFZ03, Gha00, Ora05] optimiert. Sie sind daher für die Realisierung eines migrierbaren und schnellschreibenden Verzeichnisdienstes nicht geeignet. Es muss eine eigene Lösung implementiert werden, die aber LDAP-kompatibel sein soll.

Für die Strukturierung der Einträge in einem LDAP-kompatiblen Verzeichnisdienst sollten ebenfalls standardisierte Datenstrukturen verwendet werden. Im Abschnitt 4.3.5 wurden die Standards der Agentenwelt FIPA [Fou07] und MASIF [The00] schon vorgestellt. Im FIPA-Standard *FIPA Agent Software Integration Specification* [Fou01a] werden u.a. Formate zur Beschreibung von Anwendungsdiensten vorgestellt, die zur Integration von Nicht-Agentensoftware in MAS genutzt werden können. In der implementierten Lösung sollen diese standardisierten Formate zur Beschreibung von Anwendungsdiensten Anwendung

finden.

8.1.2 Vorbetrachtung und Analyse zu Region verwalten und adaptieren

8.1.2.1 Zentrale vs. dezentrale Lösung

Ein Verwaltungsdienst für Anwendungsdienstinformationen kann entweder als verteilte oder als zentrale Lösung realisiert werden. Bei der Designentscheidung spielen Einsatzbedingungen eine große Rolle. Die Beschränkung des zu verwaltenden Bereichs auf eine Region, also ein lokales logisches Netzwerk, begrenzt auch die Anzahl der zu verwaltenden Knoten und damit der zu verwaltenden Anwendungsdienstinformationen. Daher spielt für die Verwaltung der Anwendungsdienstinformationen innerhalb einer Region die Mengenskalierbarkeit nur eine untergeordnete Rolle.

Eine verteilte Lösung zur Informationsverwaltung wäre denkbar, bietet aber bei Netzwerkdynamik Nachteile bezüglich Netzwerklast und Informationskonsistenz. Ihre Vorteile, die gegenüber einer zentralen Lösung in einer ausfallsichereren Verwaltung der Informationen und geringerer Speicherbelastung der Netzwerkknoten liegt, kann eine solche Lösung jedoch bei der geringeren Dynamik und der relativ geringen Anzahl der Anwendungsdienste in einer Region nicht ausspielen.

Eine zentrale Lösung zur Verwaltung der Anwendungsdienste bietet sich unter den angenommenen Voraussetzungen an. Durch sie entfällt der Kommunikationsaufwand für die Konsistenthaltung der Dienstinformationen gegenüber einer verteilten Lösung, was wiederum das Netzwerk entlastet. Die relativ geringe Anzahl an Anwendungsdienstinformationen in einer Region sollte auch keine hohen Anforderungen an den Speicherbedarf des Netzwerkknotens stellen, der den Verwaltungsdienst hostet. Bei einer zentralen Lösung müssen allerdings Maßnahmen zur Sicherung der zentralen Informationsverwaltung ergriffen werden, die einen Single-Point-of-Failure darstellen kann.

8.1.2.2 Sicherungsmaßnahmen und Selbstadaption

Ein Verwaltungs- und Verzeichnisdienst, auch ein zentral verwalteter, soll möglichst jederzeit verfügbar sein. Dies bedeutet, dass er weder überlastet werden noch ausfallen darf. Bei Störungen, wie sie in komplexen verteilten Systemen immer wieder auftreten, darf sich der Ausfall einer Komponente nur geringfügig auf die Gesamtfunktionalität des Systems auswirken[4], d.h. die Komponenten dürfen dafür nur lose gekoppelt sein. Es muss daher sichergestellt werden, dass

[4] Vergleiche dazu auch in Abschnitt 4.2.3 das Thema Kopplungsgrad im Zusammenhang mit den Parametern von Peer-to-Peer-Systemen.

die Ausfallwahrscheinlichkeit einer wichtigen Komponente, wie einem zentralen Verzeichnisdienst, gering gehalten wird.

Die absolute Anzahl der verwalteten Anwendungsdienste und die Frequenz ihrer Änderungen/Aktualsierungen und die damit verbundene Netzwerklast sollte, wie im vorherigen Abschnitt beschrieben, für die Belastung eines zentralen Verzeichnisdienstes keine Rolle spielen. Es kann aber passieren, dass die durch die An- und Abmeldevorgänge entstehende Rechenlast auf dem Host eine nicht unerhebliche Größe annehmen kann. Als Gegenmaßnahme dazu sollte immer der rechenstärkste Netzwerkknoten als Host für den Verwaltungsdienst dienen, der gleichzeitig genügend Speicherplatz zur Verfügung stellen kann. Da sich Speicherplatz und -auslastung sowie Rechenauslastung über die Zeit ändern können, muss eine ständige Überprüfung der Knoten auf diese Leistungswerte[5] stattfinden. Fällt der Host des Verwaltungsdienstes leistungsmäßig auf Dauer zu weit gegenüber anderen Knoten des Netzwerkes zurück oder betritt ein neuer Knoten mit besseren Leistungswerten das Netzwerk, so sollte dieser den Verwaltungsdienst im laufenden Betrieb übernehmen können.

Der Ausfall eines zentralen Dienstes stellt in jedem System einen heiklen Fehlerfall dar, der sich nicht immer verhindern lässt. In einem dynamischen Netzwerk können potentiell alle Rechner entweder aus ihrer Region wegmigrieren, heruntergefahren werden oder einfach ausfallen. Allerdings kann die Wahrscheinlichkeit, dass ein bestimmter Rechner in einem begrenzten Netzwerk ausfällt, gering gehalten werden. Daher sollte im dynamischen Netzwerk immer eine möglichst stabile und zuverlässige Plattform einen solchen Dienst hosten. Die Zuverlässigkeit eines Hostes bezieht sich dabei nicht nur auf seine Leistungsfähigkeit, sondern auch auf seine zuverlässige Erreichbarkeit im Netzwerk. In einem dynamischen Netzwerk, welches aus dynamischen, hochdynamischen, aber auch aus festen Plattformen besteht, wäre die Wahl einer festen Plattform als Host für den Verwaltungsdienst die beste, da die Wahrscheinlichkeit, dass diese aus dem Netzwerk migriert, ausfällt. Auch sind die Onlinezeiten von fest mit dem Netzwerk verbundenen Rechnern meist höher als bei mobilen Rechnern bzw. sind sie sogar dauerhaft online. Die Wahrscheinlichkeit, dass ein Rechner, der schon lange online ist, plötzlich offline geht, ist gering. So kann die Dauer der Mitgliedschaft eines Knotens im logischen Netzwerk[6] als Wahrscheinlichkeitskriterium für seine weitere Anwesenheit und damit als Kriterium für die zuverlässige Erreichbarkeit gewertet werden. Dieser Aspekt sollte bei der Wahl des Hostes für den Verwaltungsdienst berücksichtigt werden.

Um auf eine Überlastung oder einen Ausfall des zentralen Verwaltungsdienstes reagieren zu können, sollte das logische Netzwerk also mindestens den Verwal-

[5] In QuickLinkNet werden die Leistungswerte von der Komponente PerformanceAnalyer des Dienstes QuickLink, wie in Abschnitt 7.6 beschrieben, gemessen und geliefert.

[6] Dieser Wert wird u.a. vom PerformanceAnalyser des Infrastrukturdienstes QuickLink gemessen und ServiceJuggler zur Verfügung gestellt, siehe Abschnitt 7.6.

tungsdienst für Anwendungsdienstinformationen auf andere Knoten migrieren und sich selbst so an dynamische Veränderungen adaptieren können. Daraus ergeben sich für den Verzeichnisdienst zusätzliche Anforderungen: Er muss schnell migriert und nach einer erfolgten Migration schnell gestartet werden können.

8.1.3 Vorbetrachtung und Analyse zu lokale und globale Sicht intelligent verbinden

Eine Aufgabe ServiceJugglers ist die Verwaltung von Anwendungsdienstinformationen einer Region. Eine weitere seiner Aufgaben ist es, die Verbindung der Region zum globalen System herzustellen. Dazu muss ServiceJuggler entscheiden, wann es welche Informationen über Anwendungsdienste in das globale System weiterleitet, um sie auch für mobile Agenten und andere verteilte Anwendungen fremder Regionen erreichbar zu machen.

Im globalen System QuickLinkNets, welches durch APLICOOVER abgebildet wird, herrschen andere Umgebungsbedingungen als auf der lokalen Netzwerkebene ServiceJugglers. So liegt die abbildbare Netzwerkdynamik wegen der globalen Netzwerkausdehnung auf einem niedrigeren Niveau (vgl. Abschnitt 5.2.1), gleichzeitig muss jedoch die globale Mengenskalierung aller potentiell registrierbaren Anwendungsdienste sichergestellt werden. Ein Verwaltungsdienst für Anwendungsdienste im lokalen logischen Netzwerk, wie ServiceJuggler es darstellt, muss daher die unterschiedlichen Bedingungen beachten.

Eine Reduzierung der Menge der anzumeldenden Anwendungsdienste im globalen System ist in Bezug auf die globale Skalierbarkeit sehr hilfreich, da so die erforderliche Nachrichtenmenge im globalen System verringert wird. Die so eingesparte Bandbreite kann u.a. zu Erhöhung der abbildbaren Dynamik eingesetzt werden. Eine intelligente Lösung zur Reduzierung der Zahl der Anwendungsdienste sollte daher keine Option, sondern ein MUSS sein. Ferner ergibt sich aus der geringeren Netzwerkdynamik im globalen System die Notwendigkeit, nur solche Anwendungsdienste verfügbar zu machen, die schon eine gewisse Stabilität, also Langlebigkeit, innerhalb einer Region nachgewiesen haben. Ein Verwaltungsdienst wie ServiceJuggler muss dies berücksichtigen.

8.1.4 Aufbau und Architektur des Infrastrukturddienstes ServiceJuggler

ServiceJuggler ist modular aufgebaut. Es besteht, wie in Abbildung 8.2 dargestellt, aus den drei Softwarekomponenten:

- *ServiceManager*,
- *ServiceRegistration* und
- *ServiceRegistry*.

Über die *Interfaces* können andere Dienste und verteilte Anwendungen wie MAS und MA die Softwarekomponenten von außen ansprechen.

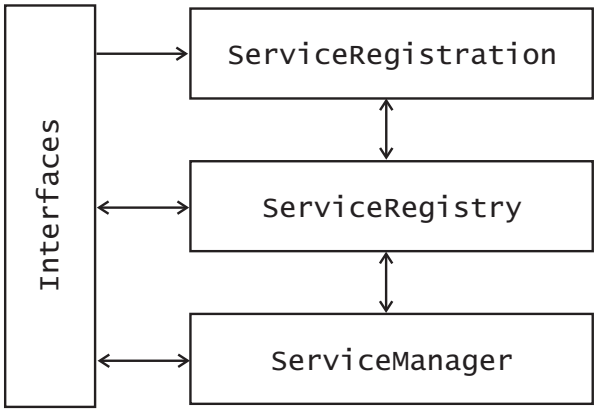

Abbildung 8.2: Die Architektur von ServiceJuggler

Das zentrale Softwaremodul *ServiceManager* realisiert alle die Aufgaben ServiceJugglers, die auf jeder Plattform lokal erfüllt werden müssen. Dazu gehören zum einen alle Funktionen, die in die Überwachung des lokalen Netzwerkes involviert sind. So detektiert der ServiceManager auch den Ausfall von anderen Knoten und löst das Löschen aller Anwendungsdienste, welche der ausgefallene Knoten in der Region registriert hatte, aus. Zum anderen erfüllen die ServiceManager aller Plattformen im lokalen Netzwerk im Zusammenspiel die Adaption der Region an sich ändernde Umgebungsbedingungen.

Für die spezielle Aufgabe, die Anwendungsdienste einer Region zu verwalten, bedient sich der ServiceManager des Moduls *ServiceRegistry*. Es wird immer nur auf einer Plattform in einer Region gestartet, wodurch die betreffende Plattform zum Regions-Manager aufsteigt[7]. Der ServiceRegistry kapselt demnach die Funktion des in der Region zentral geführten Verzeichnisdienstes für Anwendungsdienstinformationen. Ferner übernimmt der ServiceRegistry die intelligente Übermittlung dieser Informationen an den Infrastrukturdienst APLICOOVER, welcher die Verbindung der Region zum globalen Netzwerk herstellt. Für die Suche nach Anwendungsdiensten stellt der ServiceRegistry eine Schnittstelle bereit, die über Java-RMI bekannt gemacht wird.

Zur intraregionalen Kommunikation mit dem zentral auf einem Knoten geführten ServiceRegistry dient das Modul *ServiceRegistration*, welches die Registrierung und Deregistrierung von Anwendungsdiensten am ServiceRegistry realisiert. Es muß daher auf jeder Plattform, auf der verteilte Anwendungen wie

[7] Der Regions-Manager ist mit dem Start des ServiceRegistry regional arbeitsfähig, wird aber erst durch die zusätzliche Einbindung von APLICOOVER befähigt, überregionale Informationen aus anderen Regionen anbieten zu können.

MAS Anwendungsdienstinformationen austauschen wollen, gestartet sein. Da die Suche nach Anwendungsdiensten nicht über dieses Modul geschieht, muss es auch nicht gestartet werden, wenn der betreffende Knoten keine eigenen Anwendungsdienste publizieren möchte. In den folgenden Abschnitten werden die drei Module detailliert beschrieben.

8.2 Die Softwarekomponente ServiceManager

8.2.1 Die Aufgaben des ServiceManagers

Die wichtigste und zentrale Softwarekomponente ServiceJugglers ist der ServiceManager. Ihm kommen die meisten und die komplexesten Aufgaben in ServiceJuggler zu. Dies sind, abgeleitet aus den Vorbetrachtungen zu ServiceJuggler in Abschnitt 8.1, im Einzelnen:

- situationsgebundene Verwaltung (Start, Beendigung und Migration) aller Komponenten ServiceJugglers auf der lokalen Plattform,
- ständige Überwachung der Prioritätswerte aller QuickLink-Knoten im Netzwerk,
- Wahl des Regions-Managers und Starten von APLICOOVER und
- die ständige Überwachung der QuickLink-Knoten auf deren Eignung als Regions-Manager.

Der ServiceManager ist als Java-Thread implementiert, welcher in regelmäßigen Abständen die Methoden zur Überwachung der lokalen ServiceJuggler-Instanz ausführt und dabei ihre Rolle in der Region und ihre Leistungseigenschaften überprüft. Die Leistungseigenschaften erhält ServiceJuggler von QuickLink, welches die Werte der eigenen Plattform in regelmäßigen Abständen misst[8], die Werte der anderen Plattformen in der Region durch die Zyklusnachrichten der QuickLink-Knoten[9] erhält und diese in seiner Priority-List abspeichert.

8.2.2 Vorgänge und Funktionsweise

8.2.2.1 Die Situationen und Verhaltensweisen des ServiceManagers

Ein ServiceManager kann sich in Abhängigkeit von der Rolle seiner Plattform innerhalb der Region in verschiedenen Situationen befinden. Je nach vorherrschender Situation nimmt er einen anderen Zustand an und verändert auch sein Verhalten. So muss der ServiceManager des Regions-Managers bedeutend mehr

[8] Vergleiche dazu auch Abschnitt 7.6.
[9] Vergleiche dazu auch Abschnitt 7.3.1.4.

Aufgaben erledigen als seine Partner-Instanzen auf den normalen Plattformen der Region. Die möglichen Zustände eines ServiceManagers sind in Abbildung 8.3 dargestellt.

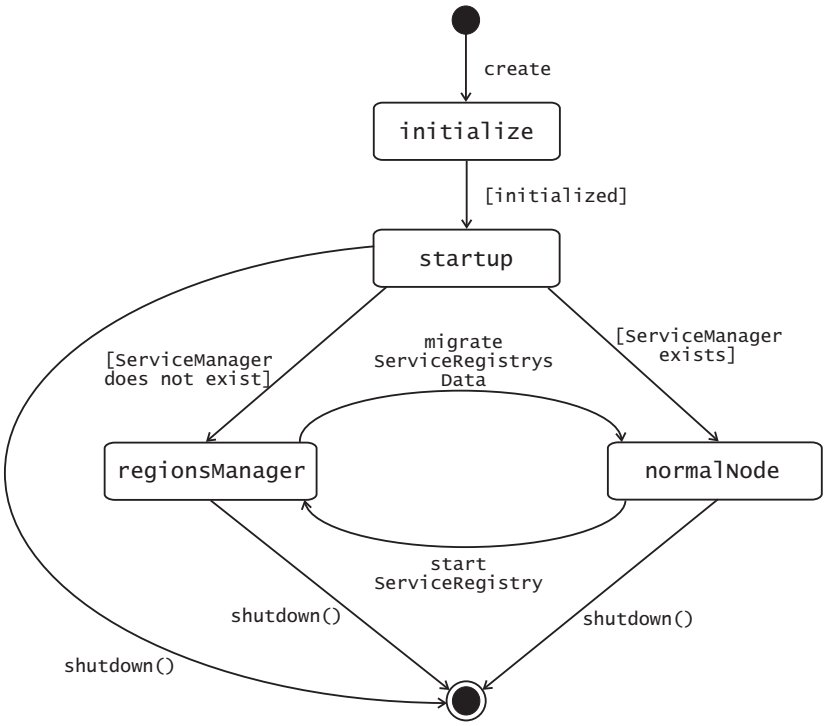

Abbildung 8.3: Das Zustandsdiagramm des ServiceManagers

Beim Start einer ServiceJuggler-Instanz auf einem QuickLink-Knoten wird zuerst der ServiceManager gestartet. Ihm werden beim Start alle benötigten Interfaces und der RMI-Namingserver als Parameter übergeben. Der ServiceManager befindet sich nun im Zustand *initialize*. Er liest nun seine Konfigurationsdatei mit den Initialwerten aus, legt seinerseits alle benötigten Objekte an und initalisiert diese. Ferner werden seine Schnittstellen am RMI-Namingserver registriert. In dieser Phase wird auch das Modul *ServiceRegistration* gestartet. Sind diese Vorgänge abgeschlossen, ist der ServiceManager einsatzbereit und geht in den Zustand *startup* über. In diesem Zustand muss sich der ServiceManager entscheiden, welche Rolle er im Netzwerk annimmt. Die zur Entscheidungsfindung nötigen Vorgänge werden in den nächsten Abschnitten 8.2.2.2 und 8.2.2.3 beschrieben.

Nimmt der ServiceManager die Rolle als Regions-Manager an, geht er in den Zustand *regionsManager* über. Dazu muss er, wie in Abschnitt 8.2.2.2 beschrieben wird, einen ServiceRegistry starten und erweiterte Managementfunktionen erfüllen. Die wichtigste Funktion ist dabei die leistungsbezogene Überwachung

der QuickLink-Knoten der Region und das eventuelle Auslösen der Migration des ServiceRegistry auf einen potenteren Knoten.

Nimmt der ServiceManager dagegen die Rolle eines normalen QuickLink-Knotens an, geht er in den Zustand *normalNode* über. Seine Funktionen beschränken sich dann auf die Überwachung des eigenen Knotens und die Kontrolle des ordnungsgemäßen Zustandes der Region. Dazu überwacht der ServiceManager das Vorhandensein eines Regions-Managers und die Performance (Prioritätswerte) aller anderen QuickLink-Knoten. Eigene Aktionen innerhalb der Region löst der ServiceManager in diesem Zustand nur aus, wenn der Regions-Manager ausfällt, also ein Fehlerfall vorliegt.

Ist ein QuickLink-Knoten auf Dauer besser als Regions-Manager geeignet, so kann dieser normale Knoten die Rolle des Regions-Managers durch Starten eines ServiceRegistry und Übernahme der Daten vom alten Regions-Manager annehmen. Die beiden Knoten tauschen dann ihre Rolle und die Zustände ihrer ServiceManager. Durch die shutdown()-Methode kann der ServiceManager aus jedem Zustand außer *initialize* beendet werden.

8.2.2.2 Starten und Bootstrapping

Nachdem der ServiceManager angelegt und initialisiert ist, geht er in den Zustand *startup* über. In diesem Zustand hat er noch keine Informationen über die Region und die anderen QuickLink-Knoten im Netzwerk. Daher bezieht der ServiceManager zuerst von seiner lokalen QuickLink-Schicht die Information, ob eine der in QuickLink bekannten Plattformen einen ServiceRegistry betreibt und somit die Rolle des Regions-Managers einnimmt. Ist dies der Fall, geht der ServiceManager sofort in den Zustand *normalNode* über. Falls die neue Plattform eine weitaus höhere Perfomance aufweist als der Regions-Manager, sollte sie die Rolle des Regions-Managers übernehmen. Die nötigen Performanceinformationen in Form von Prioritätswerten kann jeder ServiceManager von seiner QuickLink-Schicht beziehen. Die Entscheidung darüber, ob die neue Plattform zum Regions-Manager werden soll und das darauf folgende Auslösen des Übergangs des ServiceRegistry an die neue Plattform, obliegt aber einzig dem ServiceManager des schon existierenden Regions-Managers.

Kann über QuickLink kein Regions-Manager ermittelt werden, so ist die Region in einer Phase der Neufindung eines solchen. Diese Situation tritt aber nur beim Entstehen einer Region oder beim unvorhergesehenen Ausfall des Regions-Managers auf. Da der Zustand, dass ein QuickLink-Netzwerk keinen Regions-Manager hat, von allen QuickLink-Instanzen und damit auch allen ServiceJuggler-Instanzen in der Region gleichzeitig wahrgenommen wird, läuft die nun folgende Knotenwahl ebenfalls simultan auf jedem Knoten ab. Die Knotenwahl wird im nächsten Abschnitt beschrieben.

8.2.2.3 Auswahl des Regions-Managers

Die Auswahl des Regions-Manager erfolgt in einer Routine des ServiceManagers. Diese holt sich dazu die Performancewerte[10] aller bekannten QuickLink-Knoten aus der QuickLink-Schicht und vergleicht diese mit denen des eigenen Knotens. Zur Wahl selbst wird in der aktuellen Implementierung nur ein Prioritätswert von verschiedenen verwendet. In späteren Ausbaustufen wird die Auswertung mehrerer Prioritätswerte bzw. die Kombination mehrerer Werte zu einer kummulierten Größe möglich sein, um die Charakteristik des Auswahlalgorithmus und damit das Verhalten der Region an andere verteilte Anwendungstypen als einem MAS anpassen zu können. In der aktuellen prototypischen Implementierung muss der zur Regions-Manager-Wahl verwendete Prioritätswert vor dem Start in der Konfigurationsdatei ServiceJugglers festgelegt werden. Alle ServiceJuggler-Instanzen müssen, wie in QuickLink, die gleichen Konfigurationseinstellungen bzgl. der zur Regions-Manager-Wahl verwendeten Parameter besitzen. Der Knoten mit dem im Vergleich besten absoluten Prioritätswert gewinnt die Wahl und wird Regions-Manager. Falls mehrere Knoten die gleichen Performancewerte besitzen, wird unter den gleichwertigen Knoten der mit der zahlenmäßig kleinsten IP-Adresse ausgewählt. Somit ist immer eine eindeutige Auswahl sichergestellt, auch wenn der Auswahlalgorithmus auf jedem Knoten innerhalb der Region lokal und simultan abläuft. Die Auswahlroutine kann aus drei Situationen heraus aufgerufen werden, nämlich

- beim Starten von ServiceJuggler,
- beim Ausfall eines bestehenden Regions-Managers oder
- im laufenden Betrieb beim Vorliegen besserer Performanceeigenschaften eines anderen QuickLink-Knotens.

Beim Starten des ServiceManagers von ServiceJuggler wird überprüft, ob schon ein Regions-Manager existiert. Wenn dies nicht der Fall ist, wird die Auswahlroutine angeworfen. Im Ergebnis wird der Knoten mit dem höchsten Performancewert der Regions-Manager. Ist nur ein Knoten im Netzwerk, so wird dieser automatisch zum Regions-Manager gewählt.

Beim Ausfall eines bestehenden Regions-Manager wird ebenso wie beim Start verfahren. Im Zustandsdiagramm des ServiceManagers in Abbildung 8.3 ist diese Situation als Übergang vom Zustand *normalNode* in den Zustand *startup* dargestellt.

Eine Abart der Auswahlroutine kommt in der dritten Situation zum Einsatz, wenn schon ein Regions-Manager existiert. In diesem Fall erfolgt die Überprüfung der Prioritätswerte und Auswahl des geeignetsten Knotens nur noch vom

[10] Für die Wahl des Regions-Managers werden die Prioritätswerte verwendet, welche, wie im Kapitel 7 (QuickLink) beschrieben, aus den Performancewerten errechnet werden.

ServiceManager des Regions-Managers. Im Gegensatz zur simultan erfolgenden Auswahl, in dem einfach immer der höchste Wert entscheidet, werden an die Neuauswahl eines Regions-Managers höhere Anforderungen gestellt. Da bei der Neuauswahl eines Knotens die Funktionalität des alten Regions-Managers auf den neuen übergeht, wird nach dem Übergang die Performance des neuen Regions-Managers sinken und die des ehemaligen steigen. Um ein Aufschwingen des Systems und einen damit verbundenen ständigen Wechsel der Managerrolle durch diese Selbstbeeinflussung zu verhindern, müssen Vorkehrungen getroffen werden. Daher ist diese Auswahlroutine als diskreter Schwellwertschalter mit Hysterese und zusätzlicher Zeitverzögerung implementiert. So muss der zur Wahl verwendete Prioritätswert eines neuen Knotens p_n über eine Stabilitätszeitspanne t_s um den Hysteresefaktor h größer sein als der Wert des alten Regions-Managers p_o, damit der alte Regions-Manager den neuen Knoten zum Regions-Manager wählt. Der folgende Algorithmus in Pseudocode verdeutlicht die Arbeitsweise der Auswahlroutine:

Require: $-128 \leq (p_n, p_o) \leq 127$
Require: runState = true

```
1:  while runState do
2:    if p_n > h * p_o then
3:      take startTime
4:      boolean keepRegionManager := false
5:      while startTime + t_s < currentTime do
6:        if p_n ≤ h * p_o then
7:          keepRegionManager := true
8:          BREAK
9:        end if
10:       sleep for 1 sec
11:     end while
12:     if keepRegionManager = false then
13:       turn over role "Regions-Manager" to the new node
14:     end if
15:   end if
16:   sleep for 10 sec
17: end while
```

8.2.2.4 Starten eines ServiceRegistry und APLICOOVERs

Wird ein neuer Knoten zum Regions-Manager gewählt, so muss er zur Annahme dieser Rolle einen zusätzlichen Dienst in der Region leisten, nämlich die Verwaltung der Anwendungsdienstinformationen. Dazu muss er zuerst das Modul *ServiceRegistry* starten, dessen Aufbau und Wirkungsweise im Abschnitt 8.4 nä-

her beschrieben wird. Der Start eines ServiceRegistry kann wieder aus den drei aus dem vorherigen Abschnitt bekannten Situationen heraus auftreten:

- beim Starten von ServiceJuggler,
- beim Ausfall eines bestehenden Regions-Managers oder
- im laufenden Betrieb beim Vorliegen besserer Performanceeigenschaften eines anderen QuickLink-Knotens.

Beim Start des ServiceRegistry werden alle notwendigen Klassen angelegt, initialisiert und die für die Kommunikation notwendigen Schnittstellen am Java RMI-Namingserver angemeldet. Während der ersten beiden Situationen ist der *Startup* an dieser Stelle beendet. Nach dieser Phase ist der ServiceRegistry arbeitsbereit und die verteilten Anwendungen der Region können ihre Anwendungsdienste registrieren.

Wird ein normaler QuickLink-Knoten im laufenden Betrieb vom bis dahin aktuellen Regions-Manager zum neuen Regions-Manager gewählt, so findet nach dessen Startup-Phase zusätzlich das Übertragen der im Verzeichnisdienst des ServiceRegistry bestehenden Daten an den neuen ServiceRegistry statt. Sobald die Datenübertragung ausgelöst wird, wird auch der Status des QuickLink-Knotens, der den Regions-Manager darstellt, in QuickLink gewechselt und über eine Aktualisierungsnachricht verbreitet. Über diese Aktualisierung werden alle QuickLink-Knoten innerhalb der Region umgehend über den Wechsel des Regions-Managers informiert. Sobald der Datentransfer abgeschlossen ist und der alte Regions-Manager über seine QuickLink-Schicht vom Statuswechsel informiert wurde, fährt er seinen nun nicht mehr benötigten ServiceRegistry herunter. Sein ServiceManager geht in den Zustand *normalNode* über und arbeitet nunmehr nur noch seine Funktionen als normales Regionsmitglied ab.

Der ServiceManager des neuen Regions-Managers geht in den Zustand *regionsManager* über und nimmt nun die erweiterten Funktionen, wie die Überwachung der Prioritätswerte aller Knoten und die eventuelle Neuwahl des Regions-Managers, wahr. Der ServiceRegistry fängt nun seinerseits an, seine Managementfunktionen bzgl. der Anwendungsdienste wahrzunehmen. Diese werden in Abschnitt 8.4 erläutert. Zum Abschluss startet der ServiceRegistry die Komponente APLICOOVER, um die Einbindung der Region in das globale Netzwerk sicherzustellen.

8.2.2.5 Ausfall des Regions-Managers

Der Ausfall des Regions-Managers ist ein schwerwiegender Fehlerfall, der zur Isolation der Region und zum Verlust des Regionsprofils führt. Da ein solcher Ausfall zwar unwahrscheinlich, aber in dynamischen Netzwerken und mit mobilen Endgeräten nicht vollständig vermeidbar ist, muss ServiceJuggler auf diese

Situation reagieren können. Diese Aufgabe kommt dem ServiceManager zu. Da beim Ausfall der wichtigste Knoten der Region, der Regions-Manager, betroffen ist, kann dessen ServiceManager-Komponente nicht steuernd eingreifen. Der Ausfall muss daher von den ServiceManagern der anderen Regionsmitglieder, die sich zu der Zeit im Zustand *normalNode* befinden, gemanagt werden.

Die Detektion eines ausgefallenen Knotens wird zuerst von der QuickLink-Schicht entdeckt, welche das grundsätzliche lokale logische Netzwerk zusammenhält. Tritt diese Situation ein, so benachrichtigen die QuickLink-Schichten der Regionsmitglieder über einen *Push*-Mechanismus sofort ihre ServiceJuggler-Schichten. Dadurch können alle ServiceManager-Instanzen der Region gleichzeitig in den Zustand *startup* übergehen und einen neuen Regions-Manager wählen. Das genaue Vorgehen dabei wurde schon in den Abschnitten 8.2.2.3 und 8.2.2.4 erläutert. Alle Anwendungsdienste müssen anschließend neu registriert werden. Ebenfalls muss die Region neu in das globale Netzwerk eingebunden werden.

8.3 Die Softwarekomponente ServiceRegistration

8.3.1 Aufgaben und Funktionsweise

Die ServiceRegistration ist das zweite grundsätzliche Modul ServiceJugglers, welches auf jedem Knoten gestartet sein muss, der Anwendungsdienste publizieren möchte. Es wird beim Start von ServiceJuggler, in Abhängigkeit eines Schalterwertes aus der Konfigurationsdatei ServiceJugglers, durch den ServiceManager automatisch gestartet und über die Einbindung seiner Schnittstellen in den Java RMI-Namingservice verfügbar gemacht. Das Modul ServiceRegistration hat die Aufgabe, externe verteilte Anwendungen wie MAS oder MA mit einer standardisierten Schnittstelle zu versorgen, mit welcher diese

- Anwendungsdienste registrieren und
- Anwendungsdienste deregistrieren

können. Beim Start der ServiceRegistration wird eine Routine zum automatischen Suchen von Diensten ausgelöst, welche die eingebundenen verteilten Anwendungen zur Übergabe ihrer zu veröffentlichenden Dienste auffordert. Diese Routine kann bei Bedarf auch noch später während des normalen Betriebes ausgelöst werden. Danach verhält sich der ServiceRegistry passiv, d.h. er tritt nur in Aktion, wenn er von einer anderen Anwendung angesprochen wird.

8.3.2 Aufbau der Dienstbeschreibungen

Die ServiceRegistration bietet eine standardisierte Schnittstelle zum Anmelden und Abmelden von Diensten in der Region. Damit alle verteilten Anwendungen

die Informationen über Anwendungsdienste verstehen können, wird nun noch eine standardisierte Struktur zur Beschreibung der Dienste benötigt. Die Struktur der in QuickLinkNet eingesetzten Dienstbeschreibung entspricht dem FIPA-Standard *Agent Software Integration Specification - FIPA XC00079* [Fou01a], Kapitel 4, Absatz 4.1 "Object Descriptions", der im selbigen Standard zur Beschreibung aller Dienste empfohlen wird. Im Prototypen QuickLinkNet wird diese Art der Dienstbeschreibung aber nur zur Beschreibung von Anwendungsdiensten eingesetzt. Die folgende Tabelle 8.1 gibt einen Überblick über die verwendete Datenstruktur.

Attribut	Notwendigkeit	Datentyp
name	zwingend	String
type	zwingend	String
ontology	optional	Liste von Strings
protocol	optional	Liste von Strings
properties	optional	Liste von Strings
communication-properties	zwingend	Liste von communication-properties

Tabelle 8.1: Struktur der Dienstbeschreibung nach FIPA XC00079 [Fou01a], wie sie in QuickLinkNet verwendet wird

Die einzelnen Attribute der Datenstruktur haben dabei folgende Bedeutungen:

name Der Name des Dienstes.

type Der Typ des Dienstes.

ontology Eine Liste von Ontologienamen, die der Dienst unterstützt.

protocol Eine Liste von Interaktionsprotokollen, die der Dienst unterstützt.

properties Eine Liste von Eigenschaften, welche den Dienst beschreibt bzw. von anderen unterscheidet.

communication-properties Eine Liste von Kommunikationsmethoden, die der Dienst unterstützt.

Während die Attribute *name, type, protocol, ontology* und *properties* der Dienstbeschreibung verteilten Anwendungen als Suchkriterien dienen und bei Suchanfragen als Parameter übergeben werden können, stellt das Attribut *communication-properties* der Dienstbeschreibung immer den Ausgabewert einer Antwort auf eine Suchanfrage dar. Die *communication-properties* sind ebenfalls im erwähnten Standard spezifiziert. Tabelle 8.2 gibt einen Überblick über die verwendete Datenstruktur.

Attribut	Notwendigkeit	Datentyp
net-protocol	zwingend	String
address	zwingend	URL
message-body-format	zwingend	String
message-body-encoding	zwingend	String

Tabelle 8.2: Die Struktur der Rückgabeparameter der Antwort auf eine Suchanfrage nach dem Standard FIPA XC00079 [Fou01a], wie sie in QuickLinkNet verwendet wird

Die Attribute der Antwort haben dabei folgende Bedeutung:

net-protocol Das Netzwerkprotokoll, über das der Dienst ansprechbar ist (z.B: SMTP, HTTP, IIOP).

address Die Zugangsadresse zum Dienst.

message-body-format Das zu verwendende Format, um eine Nachricht an den Dienst zu senden (z.B. FIPA-String-ACL).

message-body-encoding Die zu verwendende Nachrichtenkodierung für eine Anfrage an den Dienst.

Durch die Verwendung der standardisierten Dienstbeschreibungen nach FIPA kann QuickLinkNet eine hohe Kompatibiltät zu MAS und anderen verteilten Anwendungen sicherstellen.

8.4 Die Softwarekomponente ServiceRegistry

Der ServiceRegistry ist das Modul ServiceJugglers, das sich ausschließlich den Anwendungsdienstinformationen der Region und deren Verwaltung widmet. Es muss dabei die folgenden Aufgaben lösen:

- Registrierungsnachrichten entgegennehmen und Anwendungsdienste registrieren,
- Deregistrierungsnachrichten entgegennehmen und Anwendungsdienste deregistrieren,
- Einträge eines QuickLink-Knotens bei dessen Verschwinden löschen und
- Anfragenachrichten nach Anwendungsdiensten entgegennehmen, die Datenbasis durchsuchen, eine Antwort generieren und zurückgeben.

- Außerdem muss noch das Regionsprofil aus den stabilen Diensten erzeugt und an APLICOOVER weitergegeben sowie ein ständiger Abgleich der lokal und global veröffentlichten Dienste geleistet werden.

Der ServiceRegistry leistet dies über seine funktionalen Bestandteile. Diese lassen sich in drei Gruppen einteilen, welche jeweils einen nach außen abgeschlossenen Bereich abbilden:

- Eine funktionale Schnittstelle als Dienstzugangspunkt für alle potentiell beteiligten Instanzen schaffen, die auf Dienste des ServiceRegistry zugreifen wollen,
- einen Verzeichnisdienst, in dem die regionalen Anwendungsdienstinformationen verwaltet werden und
- die intelligente Verwaltung der registrierten Dienste und deren Weitermeldung an APLICOOVER, der Komponente QuickLinkNets für das globale Netzwerk.

Die folgenden drei Unterabschnitte erläutern die Realisierung der drei funktionalen Bereiche näher.

8.4.1 Der Dienstzugang

Der ServiceRegistry stellt als Dienstzugangspunkt die vier Interfaces

- *Manager-ServiceRegistry*,
- *ServiceRegistration-Registry*,
- *AgentSystem-Registry* und
- *GlobalFocus-Registry*

zur Verfügung, von denen zwei nach außen zu ServiceJuggler-fremden Komponenten und zwei nach innen offeriert werden.

Nach innen wird das Interface *Manager-ServiceRegistry* für den eigenen lokalen ServiceManager des Regions-Managers bereitgestellt, über welches dieser beim Detektieren des Ausfalls eines Knotens[11] dessen gesamte angebotenen Dienste aus dem ServiceRegistry löscht.

Das zweite Interface für eigene Komponenten innerhalb ServiceJugglers mit dem Namen *ServiceRegistration-Registry* ist an die Komponente ServiceRegistration gerichtet, über welches Anwendungsdienste registriert und deregistriert

[11] Diese Information kommt aus der lokalen QuickLink-Schicht.

werden können. Der ServiceRegistration bietet diese Dienste dann seinerseits externen verteilten Anwendungen, wie MAS und MA, an.

Über das nach außen gerichtete Interface *AgentSystem-Registry* können externe verteilte Anwendungen, wie MAS und MA, nach Anwendungsdiensten, die im ServiceRegistry gespeichert sind, suchen.

Das letzte Interface, *GlobalFocus-Registry*, bietet der Komponente für das globale Netzwerk, in diesem Fall APLICOOVER, die Möglichkeit, selbst das Neuberechnen und Übergeben des Regionsprofils durch den ServiceRegistry anzustoßen.

Alle über diese Interfaces an den ServiceRegistry gestellten Anfragen und Eingaben werden in dem im Abschnitt 8.3.2 eingeführten FIPA-konformen Format gestellt. Da der zu verwendende Verzeichnisdienst, wie im Abschnitt 8.1.1.2 motiviert wurde, LDAP-kompatibel sein soll, werden von den Klassen des ServiceRegistry die FIPA-konformen Eingaben in LDAP-konforme Formate umgewandelt und an den Verzeichnisdienst weitergegeben. Dadurch ist die Verwendung eines beliebigen LDAP-konformen Verzeichnisdienstes möglich, auch wenn momentan eine eigene Implementierung verwendet wird.

8.4.2 Realisierung des Verzeichnisdienstes

Die LDAP-konformen Anfragen, mit dem der ServiceRegistry den Verzeichnisdienst versorgt, ermöglichen es, jeden LDAP-Server als Verzeichnisdienst verwenden zu können. Aber durch die spezielle Möglichkeit ServiceJugglers, den Regions-Manager dynamisch wählen zu können und damit den ServiceRegistry verschieben zu müssen, müssen die Inhalte des Verzeichnisdienstes migrierbar sein. Ebenso muss der Verzeichnisdienst ausgrund der möglichen Dynamik der Anwendungsdienste und Plattformen das schnelle Editieren (Schreiben und Löschen) von Einträgen unterstützen. In Abschnitt 8.1.1.2 wurde aber gezeigt, dass keine der untersuchten LDAP-Server-Implementierungen alle diese Eigenschaften unterstützt, geschweige denn daraufhin optimiert wurde. Die Lösung dafür ist eine eigene Implementierung, welche aus Java-Klassen besteht und einen LDAP-konformen Verzeichnisdienst realisiert. Einzelheiten zur Implementierung können Anhang A.3 entnommen werden. Die Inhalte dieses Verzeichnisdienstes sind einfach und mit wenig Aufwand migrierbar. Der Verzeichnisdienst selbst ist im Vergleich zu professionellen LDAP-Server-Produkten relativ leichtgewichtig implementiert und auf eine Anzahl von ca. 10000 Einträgen ausgelegt, was für die Anzahl von Anwendungsdiensten innerhalb einer Region als Maximum angenommen werden kann.

Die einzelnen Einträge im LDAP-konformen Verzeichnisdienst folgen grundsätzlich der in Abschnitt 8.3.2 vorgestellte Struktur *FIPA XC00079*-konformer Dienstbeschreibungen [Fou01a], werden aber noch durch zusätzliche Attribute ergänzt, welche der internen Verwaltung dienen. Im genannten FIPA-Standard

sind nur die Attribute *name* und *type* sowie die Rückgabewerte des Attributes *communication-properties* zwingend vorgeschrieben. Nun kommen noch verpflichtend die Attribute *host name*, *state* und *start time* hinzu. Der Name des Knotens, der den Dienst hostet, wird im Attribut *host name* festgehalten. Er dient als Suchattribut für das Auffinden von allen Diensten eines bestimmten Knotens und wird zur Bereinigung des Verzeichnisdienstes benötigt, wenn der ServiceManager den Ausfall eines QuickLink-Knotens mitgeteilt bekommt. In diesem Fall werden alle registrierten Dienste dieses Knotens sofort deregistriert. Das Attribut *state* dient dem Anzeigen des aktuellen Status des Diensteintrages bezogen auf die Veröffentlichung im globalen Netzwerk. Der Status ändert sich mit einer gewissen Reife des Dienstes, d.h. mit einem gewissen Alter und einer gewissen Stabilität innerhalb der Region. Um das Alter des Dienstes festzustellen, wird ein Timestamp beim Registrieren des Dienstes genommen und im Attribut *start time* gespeichert. Die beiden letzten Attribute werden im nächsten Abschnitt wieder auftauchen, in dem u.a. die Auswahl der Dienste zur Weitermeldung an APLICOOVER und deren Beziehung zum Regionsprofil beschrieben werden.

8.4.3 Die intelligente Verwaltung der Dienste und die Kommunikation mit APLICOOVER

Der ServiceRegistry verwaltet neben den Anwendungsdienstinformationen der Region auch die Kommunikation mit dem globalen logischen Netzwerk, mit APLICOOVER. Da sich nur Anwendungsdienste mit einer gewissen Stabilität lohnen, global veröffentlicht zu werden, müssen diese Dienste erst einmal herausgefunden werden. Der ServiceRegistry verwendet dazu eine zeitgesteuerte Routine, welche die Stabilität eines Dienstes anhand seiner Onlinezeit bemisst. Diese muss größer als die Zeit t_{stable} sein, um als stabil zu gelten. t_{stable} kann in der Konfigurationsdatei ServiceJugglers festgelegt werden. Zur Berechnung wird bei der Registrierung eines Dienstes ein Timestamp im Attribut *start time* des Diensteintrages im Verzeichnisdienst festgehalten und sein Status im Attribut *state* von `new` auf `registered` gesetzt. Der folgende Algorithmus beschreibt die Überwachungsroutine für einen regional registrierten, aber noch nicht global veröffentlichten Dienst:

Require: state = registered
Require: start time is taken
Require: runState = true
1: **while** runState **do**
2: **if** current time > start time + t_{stable} **then**
3: sleep for 1 sec
4: **else**

```
5:        state := stable
6:        update APLICOOVER
7:        BREAK
8:      end if
9:  end while
```

Ist eine gewisse Onlinezeit t_{stable} erreicht, wird der Zustand *state* des Diensteintrages auf `stable` gesetzt, wodurch er einen stabilen Status erreicht hat und an APLICOOVER weitergemeldet wird. Die folgende Abbildung 8.4 zeigt das Zustandsdiagramm eines Diensteintrages im Verzeichnisdienst des ServiceRegistry.

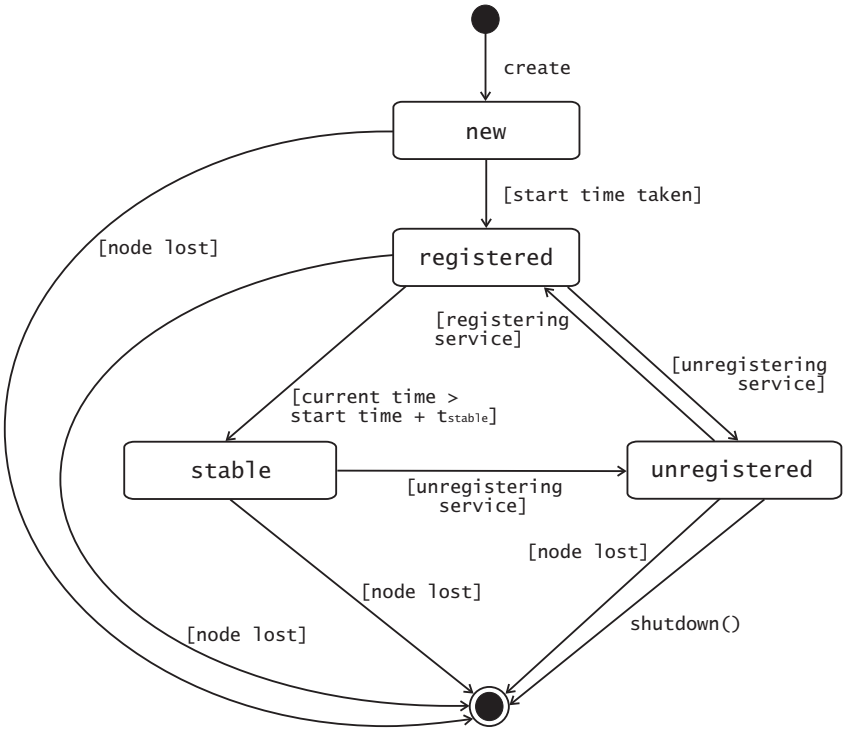

Abbildung 8.4: Die Zustände eines Diensteintrages im Verzeichnisdienst des ServiceRegistry

Wird der Dienst von seiner verteilten Anwendung deregistriert, so geht auch der Diensteintrag sofort in den Zustand `unregistered` über. Dies kann sowohl aus dem Zustand `registered`, wenn der Dienst schon in der Region verfügbar ist, als auch aus dem Zustand `stable` erfolgen. Im letzteren Fall muss der ServiceRegistry den Verlust des Dienstes auch an APLICOOVER weitermelden. Geht der den Dienst hostende Knoten unvorhergesehen verloren, so löscht der ServiceManager des Regions-Managers alle Dienste, die der betreffende Knoten im ServiceRegistry registriert hatte. Dieser Vorgang kann aus jedem der vier

Zustände des Diensteintrages erfolgen.

Alle Dienste des ServiceRegistry im Zustand *stable* werden bei der Berechnung des Regionsprofils berücksichtigt. Das Regionsprofil wird nur zu Beginn des Zutritts der Region zum globalen Netzwerk aus den vorhandenen stabilen Diensten berechnet und im weiteren Lebenzyklus der Region lediglich dann aktualisiert, wenn Änderungen bei den registrierten Diensten auftreten. Für den Zutritt der Region zum globalen Netzwerk startet der ServiceRegistry eine Instanz von APLICOOVER, wartet dann auf dessen Anfrage nach dem Regionsprofil und übergibt dieses APLICOOVER.

Zur Berechnung des Regionsprofils durchläuft der ServiceRegistry alle Einträge seines Verzeichnisdienstes und merkt sich für jede Dienstart deren Namen, Typ und die Kardinalität ihres Auftretens im Verzeichnis. Ist z.B. die Dienstart *exampleService* mit Diensttyp *application* von zehn Knoten registriert, so speichert der ServiceRegistry im Regionsprofil den Dienst *exampleService* nur einmal mit der Kardinalität 10. Sind alle registrierten Dienstarten durchlaufen und im Regionsprofil gespeichert, gibt der ServiceRegistry die einzelnen Diensteinträge des Regionsprofils sukzessive an APLICOOVER weiter, bis das gesamte Regionsprofil abgearbeitet ist. Alle Diensteinträge des Regionsprofils sind an die Knoten-IP-Adresse des Regions-Managers gebunden. Durch die Verwendung des Regionsprofils in Verbindung mit nur einer Knotenadresse kann die Menge der im globalen Netzwerk veröffentlichten Dienstbeschreibungen erheblich reduziert werden.

Tritt eine Änderung im Verzeichnisdienst des ServiceRegistry auf, so wird APLICOOVER davon benachrichtigt, um die veröffentlichten Dienstinformationen im lokalen und globalen Netzwerk konsistent zu halten. Beim Registrieren eines neuen Dienstes einer völlig neuen Dienstart im Verzeichnisdienst muss sich dieser erst einmal, wie oben beschrieben, stabilisieren, um als neuer Diensteintrag in das Regionsprofil aufgenommen und an APLICOOVER weitergemeldet zu werden. Wird ein Dienst einer schon bekannten Dienstart registriert und kommt in den Zustand *stable*, wird zuerst der betreffende Eintrag im Regionsprofil aktualisiert, d.h. seine Kardinalität wird um eins erhöht. Danach wird der gesamte aktualisierte Diensteintrag des Regionsprofils an APLICOOVER weitergeben und im globalen Netzwerk aktualisiert. Beim Deregistrieren eines Dienstes bzw. Löschen wegen Knotenausfalls wird ebenso verfahren, nur dass der Eintrag im Regionsprofil sofort, ohne zeitliche Verzögerung, vorgenommen wird. Beträgt die Kardinalität eines Diensteintrages im Regionsprofil 1 und soll diese weiter reduziert werden, wird statt der Aktualisierungsnachricht eine Löschnachricht an APLICOOVER gesendet.

Im Vorgriff auf die Beschreibung APLICOOVERs im nächsten Kapitel sei hier erwähnt, dass dieses seine Diensteinträge mittels einer *Dienst-GUID (Dienst-global unique identifier)* verwaltet. An der Schnittstelle des ServiceRegistry zu

APLICOOVER muss daher eine Übersetzung des in ServiceJuggler verwendeten Dienstnamens zur GUID des Diensteintrages in APLICOOVER stattfinden. Dies erfolgt über ein Modul APLICOOVERs namens `Utils`, welches die passenden Methoden zur Übersetzung in beide Richtungen anbietet. Da die Namen der Dienstbeschreibungen im Regionsprofil eindeutig und einzigartig sind, ist eine eindeutige Zuordnung zwischen Dienstnamen und GUID möglich.

8.5 Die Interfaces

Der Infrastrukturdienst ServiceJuggler ist modular aufgebaut. Die Module können, je nach gewünschtem bzw. benötigtem Funktionsumfang, einzeln und unabhängig betrieben werden. Deshalb bieten auch alle drei Module ServiceJugglers Interfaces an. ServiceJuggler verwendet Interfaces sowohl intern, zur Einbindung der Module untereinander, als auch nach außen als Schnittstellen zu anderen Komponenten QuickLinkNets und zu den verteilten Anwendungen. Da die nach innen gerichteten Interfaces zur Verdeutlichung der Funktionalität ServiceJugglers nicht von Belang sind[12], werden folgend nur die nach außen angebotenen Interfaces beschrieben.

Der ServiceManager bietet nach außen nur ein Interface zu QuickLink mit einer Methode an:

- Das Interface *Quicklink* enthält die Methode `deleteNode()`, um den ServiceManager über den Ausfall eines QuickLink-Knotens zu informieren. Der ServiceManager des Regions-Managers löscht daraufhin die vom ausgefallenen Knoten registrierten Anwendungsdienste aus dem Verzeichnisdienst des ServiceRegistry. Dieser aktualisiert darauf folgend gegebenenfalls das Regionsprofil und APLICOOVER.

Das Modul ServiceRegistration dient der regulären Registrierung und Deregistrierung von Anwendungsdiensten im Verzeichnisdienst des ServiceRegistry. Folglich stellt es ein Interface mit zwei Methoden zur Verfügung, welches von verteilten Anwendungen, wie MAS und MA, genutzt werden kann:

- Das Interface *AgentSystem-Registration* enthält die Methoden `register()` und `deregister()`, um externen Anwendungen die An- und Abmeldung von Anwendungsdiensten im Verzeichnisdienst zu ermöglichen.

Die Interfaces des ServiceRegistry wurden schon im Abschnitt 8.4.1 im Zusammenhang mit dem Dienstzugangspunkt zum Verzeichnisdienst behandelt. Die nach außen gerichteten Interfaces beinhalten dabei jeweils nur eine Methode:

[12] Weitere Informationen dazu können dem Anhang A.3 entnommen werden.

- *AgentSystem-Registry* enthält die Methode `RMIsuchen()`, um externen Anwendungen die Suche nach Anwendungsdiensten im Verzeichnisdienst zu ermöglichen.

- *GlobalFocus-Registry* enthält die Methode `RMIgetDomInfo()`, über die APLICOOVER nach seinem Start das Regionsprofil abrufen kann.

8.6 Zusammenfassung

Der Infrastrukturdienst ServiceJuggler bildet die mittlere Schicht im komplexen Infrastrukturdienst-Framework QuickLinkNet. Er arbeitet auf der Abstraktionsebene der Anwendungsdienste und verwaltet diese innerhalb eines lokalen, begrenzten, logischen Netzwerkes, welches als Region bezeichnet wird. Ein Knoten innerhalb einer Region übernimmt die Rolle des Regions-Manager und ist für die Verwaltung der lokal bekannten Anwendungsdienste zuständig. Diese Rolle kann dynamisch von jedem Knoten innerhalb der Region angenommen werden. Die Wahl des Regions-Managers vollzieht sich automatisch und wird simultan auf allen Knoten vollzogen. Sie beruht auf einem zuvor ausgewählten Leistungsparameter, der in der Konfigurationsdatei ServiceJugglers festgelegt wird. Es wird immer der bzgl. dieses Leistungsparameters am besten geeignete Knoten zum Regions-Manager gewählt. Dadurch wird es der Region möglich, sich bei Bedarf an das dynamische Netzwerk zu adaptieren.

In seiner Rolle als Verwalter der Anwendungsdienste übernimmt der Regions-Manager auch die Verbindung der Region mit dem globalen Teil des logischen Netzwerkes. Neben der intelligenten[13] Auswahl der an das globale Netzwerk weiterzumeldenden Dienste reduziert der Regions-Manager auch die Anzahl der zu meldenden Anwendungsdienste, indem er den Informationsinhalt seines Verzeichnisdienstes zu einem Regionsprofil verdichtet und nur mit seiner eigenen IP-Adresse koppelt. Diese Maßnahme führt zu einer zusätzlichen Verbesserung der Skalierbarkeit und zu einer Verringerung der abzubildenden Dynamik im globalen Netzwerk.

Ein weiterer Vorteil der zentralen Informationsverwaltung in der Region ist die Möglichkeit der lokalen Lastverteilung durch den Regions-Manager[14]. Bei Anfragen nach einem Dienst kann der Regions-Manager die konkreten Zuweisungen zu den Dienstinstanzen so vornehmen, dass die Anfragen über alle Dienstinstanzen

[13] Die intelligente Auswahl der Dienste kann nach verschiedenen Kriterien erfolgen und sollte sich nach der gewünschten Charakteristik des geplanten Peer-to-Peer-Systems richten. Für MAS der Stufe 2 spielen vor allem die Zuverlässigkeit der Dienstinformationen eine wichtige Rolle. Daher wurde in der prototypischen Implementierung ServiceJugglers Wert auf Stabilität und Zuverlässigkeit der Dienstinformationen gelegt.

[14] Diese Funktion ist im Prototypen z.Z. noch nicht implementiert.

in der Region gleichmäßig verteilt und lokale Überlastsituationen weitestgehend vermieden werden können.

9 Anwendungsdienstinformationen global verwalten - APLICOOVER

Dieses Kapitel behandelt den Infrastrukturdienst und die Softwarekomponente APLICOOVER, welches die dritte Infrastrukturdienstkomponente im komplexen Infrastrukturdienst *QuickLinkNet* darstellt. APLICOOVER ermöglicht die globale Verbindung von lokalen QuickLink-Regionen, stellt also die globale Sicht auf das gesamte logische Netzwerk her. Die Position der Softwarekomponente APLICOOVER in Abbildung 9.1 als oberste Schicht innerhalb QuickLinkNets spiegelt damit auch seine funktionale Bedeutung wider.

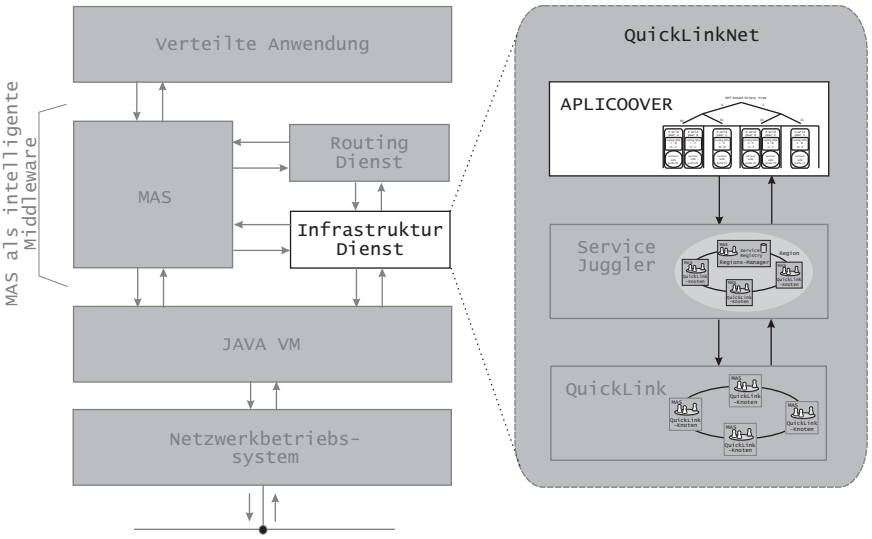

Abbildung 9.1: Der Infrastrukturdienst APLICOOVER im Framework QuickLinkNet

Das Kapitel teilt sich in acht Abschnitte. In Abschnitt 9.1 wird ein Überblick über den Infrastrukturdienst APLICOOVER gegeben sowie seine Aufgaben, die Randbedingungen seines Einsatzes und sein interner Aufbau beschrieben. Die folgenden Abschnitte 9.2 bis 9.6 beschreiben die einzelnen Komponenten APLICOOVERs detailliert. In Abschnitt 9.7 werden dann die wichtigsten Vorgänge und Funktionen APLICOOVERs beschrieben, die sich zwischen den internen Komponenten und den externen Diensten und verteilten Anwendungen abspielen. Eine Zusammenfassung des Kapitels wird in Abschnitt 9.8 gegeben.

9.1 Der Infrastrukturdienst APLICOOVER im Überblick

9.1.1 Aufgaben und Randbedingungen des logischen Netzwerkes

9.1.1.1 Aufgaben

Das globale logische Netzwerk realisiert die *globale, anwendungsdienstbezogene Sicht*[1] auf ein verteiltes System, wie es z.B. ein MAS darstellt. Dazu muss es die lokalen logischen Netzwerke auf der Ebene der Anwendungsdienstinformationen miteinander verbinden und dabei die *globale Skalierbarkeit des Gesamtsystems* sicherstellen. Das zentrale Ziel des globalen logischen Netzwerkes ist, jeden global propagierten Anwendungsdienst für verteilte Anwendungen auffindbar zu machen. Dabei muss es die Konsistenz und Aktualität der Anwendungsdienstinformationen sicherstellen, ohne die mögliche Dynamik im Netzwerk und die globale Skalierbarkeit einzuschränken.

9.1.1.2 Randbedingungen

Verbindungen Die Randbedingungen, unter denen ein logisches Netzwerk mit globaler Ausdehnung arbeiten muss, unterscheiden sich von denen eines logischen Netzwerkes mit einer Ausdehnung nur im lokalen Netzwerkbereich erheblich. Da sich ein solches globales logisches Netzwerk potentiell über das gesamte Internet ausdehnen kann, überspannt es auf der Netzwerkschicht technologisch verschiedenste Teilnetzwerke[2] mit unterschiedlichen Leistungseigenschaften wie Bandbreite und Latenzzeiten. Eine Nachricht des logischen Netzwerkes wird daher verschiedene Teilnetzwerke unterschiedlich schnell passieren. Da diese Teilnetzwerke mittels des IP-Protokolls überbrückt werden, sind die technologischen Unterschiede für das logische Netzwerk transparent, also nicht wahrnehmbar. Ferner führt die Ende-zu-Ende-Sichtweise und die Art des Routings von Paketen auf der Internetebene dazu, dass für ein bestimmtes Nachrichtenpaket ein vorgegebener Pfad durch das Internet, damit die Anzahl der zu durchlaufenden Teilnetzwerke, die Anzahl der weiterleitenden Router, die Auslastung der einzelnen Netzwerkteilstrecken und damit eine vorgegebene QoS nicht garantiert werden kann[3]. Für ein logisches Netzwerk bedeutet dies, dass es nicht garantieren kann, eine Nachricht mit einer bestimmten QoS senden zu können. Es ist daher anzunehmen, dass die Qualität und Leistungsfähigkeit der Verbindungen eines globalen logischen Netzwerkes unvorhersagbar stark schwanken wird. Unter diesen Umständen ist die Gesamtleistungsfähigkeit und Qualität der Verbindungen mit denen in einem lokalen logischen Netzwerk nicht vergleichbar. Ein logisches Netzwerk mit globaler Ausdehnung unterliegt daher einer gewissen

[1] Vgl. dazu auch Abschnitt 5.2.1 auf Seite 83.
[2] Vgl. dazu auch Abschnitt 2.2.1 auf Seite 12.
[3] Vgl. dazu auch Abschnitt 2.2.2 auf Seite 15.

Reaktionsträgheit, die sich für verteilte Anwendungen vor allem durch spürbare Latenzzeiten bemerkbar machen können.

Knoten Die Knoten eines globalen logischen Netzwerkes stellen die Verbindungspunkte von lokalen logischen Netzwerken (Regionen) zum globalen System dar. Sie sollten daher durch möglichst stabil ansprechbare Plattformen gebildet werden, die über eine ihrer Aufgabe entsprechenden Leistungsfähigkeit verfügen. Die Aufgabe der Auswahl einer geeigneten Plattform übernimmt in QuickLink-Net die Komponente ServiceJuggler. Trotzdem kann nicht ausgeschlossen werden, dass auch eine solche Plattform unvorhergesehen ausfällt. In diesem Fall muss das globale Netzwerk sicherstellen, dass die Anwendungsdienstinformationen schnellstmöglich aktualisiert werden. Aufgrund der im Vergleich zum lokalen logischen Netzwerk schlechteren Leistungseigenschaften des globalen Netzwerkes und der potentiell hohen Zahl an teilnehmenden Plattformen (globale Skalierbarkeit) kann dies zeitlich nicht im Sekundenbereich erfolgen. Wenn sichergestellt werden kann, dass die Wahrscheinlichkeit des Ausfalles eines Knotens des globalen logischen Netzwerkes gering ist und die Gesamtfunktionalität des Netzwerkes nur unerheblich beeinflusst wird, ist eine Aktualisierungsgeschwindigkeit im Bereich mehrerer Minuten akzeptabel.

9.1.2 P-Grid als geeignete Technologie für ein globales logisches Netzwerk

Es existieren heute viele Lösungen für logische Netzwerke mit globaler Ausdehnung, die der Peer-to-Peer-Architektur zugerechnet werden. Die prominentesten Vertreter sind sicherlich die File-Sharing-Systeme, von denen einige exemplarisch in Teil I Abschnitt 4.2.4 der Dissertation vorgestellt und deren Systemeigenschaften auf die Verwendbarkeit zur Verwaltung von Ressourceninformationen in einem dynamischen System hin untersucht wurden. Dabei stellte sich in Abschnitt 4.2.4.5 *P-Grid* als ein System heraus, welches die in Abschnitt 4.2.5 aufgestellten Eigenschaften

1. flacher Hierarchiegrad,
2. geringer Kopplungsgrad,
3. gute Skalierbarkeit bezüglich der Menge der Knoten und der erforderlichen Verwaltungsnachrichten bei Aktualisierungen und
4. strukturiertes System mit Indexierung

einer für Netzwerkdynamik geeigneten Overlaystruktur besitzt[4].

[4] Vgl. dazu Seite 57 und Abschnitt 4.2.4.5 auf Seite 52.

Doch dies sind nicht die einzigen Gründe für die Wahl von P-Grid als unterliegende Technologie für APLICOOVER und damit die globale Netzwerkebene QuickLinkNets. In P-Grid-basierten Netzwerken, die auf einem virtuellen verteilten Suchbaum beruhen, bilden sich während der Benutzung der Suchstruktur durch das Cachen der Suchergebnisse auf den P-Grid-Knoten direkte Verbindungen zwischen den nachfragenden und anbietenden Knoten heraus. Diese Verbindungen gehorchen einem Effekt, der aus der Soziologie bekannt ist und die Struktur menschlicher Beziehungsnetzwerke beschreibt, dem *Small-World Phänomen* [Kle99]. Es beschreibt in erster Linie ein Clustering von Menschen, die miteinander gut bekannt sind und die Beziehungen der Cluster untereinander. Demnach kennt ein Individuum nur relativ wenige Personen, mit denen es aber recht häufig interagiert. Neue Bekannte lernt man meist als Bekannte seiner eigenen Bekannten kennen. Werden neue Bekannte als interessant erkannt, so werden sie dem eigenen Cluster hinzugefügt. Mit allen Bekannten, mit denen oft interagiert wird, werden so direkte Beziehungen geknüpft. Da jedes Individuum ein eigenes Cluster besitzt, sind potentiell alle Cluster miteinander verknüpft. Im P-Grid-Netzwerk entsprechen die zwischengecachten Knoten mit ihren Ressourceninformationen den Bekannten, zu denen nach erstmaliger Suche ein direkter Link besteht. Bei Suchanfragen werden auch die Caches der benachbarten Knoten, über welche die Suchanfragen im Suchbaum erfolgt, mit durchsucht. Wird die gesuchte Ressource oft nachgefragt, so ist sie in einem Cache der umliegenden Knoten enthalten und wird entsprechend schnell gefunden. Die Suche im Suchbaum, die in P-Grid bei n Knoten und k Verbindungen pro Knoten mit [Abe01]

$$O(log\ n)$$

schon sehr effizient ist, kann so durch die direkten Links die Länge der Suchpfade auf

$$log\ (n)/log\ (k)$$

reduziert [HD05] werden. Damit sinkt der Suchaufwand noch einmal beträchtlich und kann im besten Fall mit nur einer Nachricht im Netzwerk befriedigt werden.

Schon aufgrund der bisher beschriebenen Gründe empfiehlt sich P-Grid als Technologie für das globale System QuickLinkNets. Zusätzliche Argumente für P-Grid sind seine gut verstandenen Mechanismen und Phänomene, seine hohe wissenschaftliche Akzeptanz und der ausgereifte Entwicklungstand seiner Implementierung. So existiert eine hohe Zahl an Veröffentlichungen über P-Grid selbst[5] und seine Verwendung[6] in verschiedenen verteilten Anwendungen.

[5] Die Veröffentlichungen über spezifische Themen im Zusammenhang mit P-Grid kann man [Abe07, Hau07, Dat07, Sch07] entnehmen.

[6] Einen Überblick über Anwendungen im Zusammenhang mit P-Grid kann man [A+07] entnehmen.

9.1.3 Architektur des Infrastrukturdienstes APLICOOVER

Der Infrastrukturdienst APLICOOVER baut sich aus mehreren Softwarekomponenten auf, die jeweils eine abgeschlossene Funktionalität einkapseln. Dies sind im Einzelnen die Komponenten

- *Core*,
- der *Storage*,
- der *Statistics*,
- der *Messaging* und die
- die *Interfaces*,

über welche die Kommunikation mit anderen Infrastrukturdiensten, im speziellen ServiceJuggler, und mit verteilten Anwendungen abgewickelt wird. Wie in Abbildung 9.2 illustriert, kommunizieren die Systemkomponenten APLICOOVERs wiederum mit P-Grid und kapseln seine Funktionalität ein. Da P-Grid nicht zwingend zu APLICOOVER gehört und durch ein anderes System ersetzt werden kann, ist es in der Abbildung grau dargestellt. Wie aus Abbildung 9.2 ebenfalls hervorgeht, bildet *Core* die zentrale Komponente in APLICOOVER, über die auch die komplette, nach extern gerichtete Kommunikation realisiert wird. In den folgenden Abschnitten werden die einzelnen Komponenten detailliert vorgestellt.

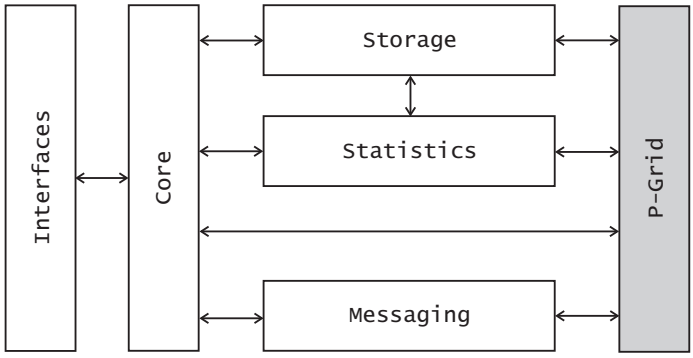

Abbildung 9.2: APLICOOVERs Softwarekomponenten im Überblick

9.2 Die Softwarekomponente Core

Die Softwarekomponente *Core* ist die zentrale Verwaltungsstelle APLICOOVERs. Sie startet, überwacht und beendet alle weiteren Komponenten und bietet

die in Abschnitt 9.6 beschriebenen Interfaces zur externen Kommunikation mit anderen Diensten und verteilten Anwendungen an. Core schirmt somit einerseits den gesamten inneren Aufbau APLICOOVERs nach außen hin ab und ermöglicht andererseits eine problemlose Anpassung des Infrastrukturdienstes. Wenn z.B. eine andere Technologie als P-Grid verwendet werden soll, kann der innere Aufbau APLICOOVERs zur Anpassung an die neue Technologie verändert werden, ohne dass sich dies auf externe Dienste und Softwaremodule auswirkt, solange die Komponente Core mit seinen Interfaces nicht geändert wird.

Core ist als einzelne Klasse implementiert. Sie implementiert alle durch die Interfaces angebotenen Methoden. Der Zugriff auf alle nach außen angebotenen Methoden erfolgt einzig über Java RMI. Dadurch wird es einerseits, auch wenn es derzeit nicht vorgesehen ist, in zukünftigen Versionen QuickLinkNets auch möglich sein, APLICOOVER und den ServiceRegistry ServiceJugglers auf verschiedenen Plattformen zu betreiben und so die Funktion des Regions-Managers auf zwei lokale Netzwerkknoten aufzuteilen. Andererseits können so alle möglichen Interessenten im lokalen QuickLink-Netzwerk auch direkt auf die Funktionen APLICOOVERs zugreifen. Im Gegensatz zur externen Kommunikation ist die Einbindung P-Grids über herkömmliche, lokale Methodenaufrufe realisiert worden.

Als Hauptkomponente obliegt es Core, die Verbindungen zum verwendeten externen Peer-to-Peer-System, in diesem Fall P-Grid, und zu den verteilten Anwendungen als potentielle Nutzer APLICOOVERs herzustellen. Nach dem Starten von Core werden APLICOOVERs Module automatisch am RMI-Registry angemeldet und so deren Zugänglichkeit für ServiceJuggler und die verteilten Dienste und Anwendungen sichergestellt. Zur weiteren Verwaltung APLICOOVERs bietet Core die Methoden `init()` und `shutdown()` über das Interface *RemoteAdministration* an, mit denen die Verbindung zum P-Grid System hergestellt bzw. abgebaut werden kann. Die Methode `init()` ruft ihrerseits nacheinander interne Methoden zum Setzen der Initialzustände und zum Übergeben der gespeicherten ehemaligen Nachbarknoten im P-Grid-Netzwerk an P-Grid auf. Erst danach wird der Bootstrapping-Vorgang zum Eintreten in das P-Grid-Netzwerk gestartet. Dieser Vorgang wird ausführlich in Abschnitt 9.7.1 beschrieben. Die Methode `shutdown()` wickelt beim Herunterfahren des Systems das Austreten aus dem globalen P-Grid-Netzwerk ab und deregistriert die Softwaremodule vom RMI-Registry.

Core implementiert alle Methoden der Interfaces *RemoteAdministration* und *RemoteServices*. Da die dahinter verborgene Funktionalität aber meist in weiteren, spezialisierten Komponenten realisiert werden, delegiert Core die Aufrufe mit dem Aufruf seiner Methoden an die entsprechenden spezialisierten Komponenten weiter. Darauf wird in den folgenden Abschnitten eingegangen.

9.3 Die Softwarekomponente Storage

Das Modul *Storage* verarbeitet in APLICOOVER alle Vorgänge, die mit der Verwaltung von Anwendungsdienstinformationen zu tun haben. Somit wird über *Storage* der gesamte Datentransfer abgewickelt, was die Bedeutung dieses Moduls unterstreicht.

9.3.1 Dienstbeschreibungen und Datentypen in APLICOOVER

P-Grid als technische Grundlage APLICOOVERs ist eigentlich als File-Sharing-System entworfen worden. Es bietet daher die Möglichkeit, verschiedene Datentypen zu verwalten. Dies geschieht nach einem ähnlichen Muster, wie WebServer Content-Typen unterscheiden, also z.B. `text/html`, `text/plain` oder `image/gif`. Die Art der nach diesem Muster verwendeten Datentypen ist in P-Grid frei wählbar und dieses Merkmal unterstützt auch APLICOOVER. Jede in APLICOOVER veröffentlichte Information muss daher an einen Typ gebunden sein, um die Vergleichbarkeit zwischen Suchanfragen und verwalteten Einträgen sicherstellen zu können. Im implementierten Prototyp werden die Vorteile dieses Leistungsmerkmals bisher noch nicht genutzt, da ServiceJuggler nur eine Art von Dienstbeschreibung (*application*) verwendet und alle von ServiceJuggler übergebenen "Dienste" in APLICOOVER Dienstbeschreibungen darstellen. Sie sind daher in APLICOOVER immer als `description` qualifiziert. In späteren Ausbaustufen QuickLinkNets kann dieses Leistungsmerkmal aber z.B. dazu genutzt werden, um Dienstnamen, die auf verschiedenen Ontologien basieren, zu verwalten und verfeinerte Suchtechniken zu implementieren. Aktuell wird aber nur der von ServiceJuggler verwendete Typ von Dienstbeschreibungen verwendet.

9.3.2 Die Hauptklasse StorageService

Die Klasse `StorageService` setzt die gesamte Funktionalität der Komponente Storage um. Sie speichert zum einen die eigenen, im globalen Netzwerk veröffentlichten Dienstinformationen und bietet Methoden zum Manipulieren dieser an. So implementiert `StorageService` alle Methoden zum Veröffentlichen, Aktualisieren und Löschen von Dienstinformationen sowie zum Löschen eines gesamten Regionsprofils. Zum anderen speichert `StorageService` alle eigenen Suchaufträge in einem lokalen Cache[7]. Zur Umsetzung der asynchronen Suchaufträge wird von `StorageService` zusätzlich jeweils ein Objekt der Klasse `SearchJob` instanziiert, auf welche im nächsten Abschnitt eingegangen wird. `StorageService` implementiert alle Funktionen, die in die Suche von Anwendungsdienstinformationen im globalen Netzwerk involviert sind. Dies sind im Einzelnen die Metho-

[7] Auf die Verwendung des Caches und seine Eigenschaften wird im Abschnitt 9.7 näher eingegangen.

den `search()`, `rangeSearch()`, `cachedSearch()` und `cachedRangeSearch()`, welche ihrerseits durch Abfragemethoden von `Core` aufgerufen werden.

Die erste Methode `search()` setzt explizit die Suche im globalen Netzwerk mit einem vollständigen Dienstnamen in Gang. Abweichend davon kann mit der Methode `rangeSearch()` nach einem Teilstring des Dienstnamens gesucht werden, da P-Grid, wie in Abschnitt 4.2.4.5 beschrieben, eine präfixerhaltende Hashfunktion zur Erzeugung der Binärschlüssel der Dienstnamen verwendet. Durch diese Methode kann eine Art unscharfe Suche umgesetzt werden. Jede dieser beiden Methoden erzeugt direkt eine Instanz der Klasse `SearchJob`, in der alle relevanten Informationen für die Suche und später, beim Erhalt der Antwort, die Suchergebnisse gespeichert werden. Alle `SearchJob`-Objekte werden mit ihrer Erzeugung im *StorageServices* Cache gespeichert. Die Objekte der beendeten Suchaufträge mit gültigen Antworten auf die Suchanfragen verbleiben dann im Cache, auch wenn die suchende Instanz sich die Antwort längst abgeholt hat und nicht mehr weiter am Ergebnis interessiert ist. Andere, später gestellte Suchanfragen können so beim Durchsuchen des Caches deutlich schneller beantwortet werden, da nicht die Suche im globalen Netzwerk angestoßen werden muss. Genau dies tun die Methoden `cachedSearch()` und `cachedRangeSearch()`, die sinngemäß den zwei vorher beschrieben Methoden entsprechen, aber nur den Cache der lokalen APLICOOVER-Instanz durchsuchen. Wenn die Suchanfrage aus dem Cache-Inhalt nicht befriedigt werden kann, geben die Methoden einen Nullwert zurück. Die anfragende Instanz muss dann wiederum eine Suchanfrage im globalen Netzwerk anstoßen.

Durch das Caching von Suchergebnissen in APLICOOVER können entfernte Regionen mit interessanten Dienstangeboten wesentlich schneller erreicht werden als mit der globalen Suche. Damit wird die Small-World-Funktionalität umgesetzt, die auch P-Grid unterstützt. Die Cache-Einträge bilden die Shortcuts im Netzwerk, über die die binärbaumartige Suchstruktur des P-Grid-Overlays umgangen werden kann. Da es in bestimmten Fällen aber unerwünscht sein kann, zuerst im Cache zu suchen, wurden die Suchmethoden für die gecachte und ungecachte Suche getrennt implementiert. Verteilte Anwendungen wie MAS und MA selbst erhalten so eine größere Flexibilität bei der Suche. Sie können selbst entscheiden, ob sie aktuellste Informationen aus dem globalen System bekommen möchten, deren Suche aber einige Zeit in Anspruch nehmen kann, oder ob sie lieber eine schnelle Antwort aus dem Cache verwenden möchten, die aber eventuell veraltet sein könnte. Da das Verhalten des Cache `StorageServices` die Qualität der Suchergebnisse stark beeinflusst, kann der Cache selbst über die Methoden `getMaxCacheTime()` ausgelesen und über `setMaxCacheTime()` parametrisiert werden. Bei zu vielen überholten Suchergebnissen im Cache kann dieser über die Methode `cleanCache()` gelöscht werden. Weitere Cache-Informationen bekommt eine interessierte Anwendung durch weitere Methoden: Die Anzahl der

Suchergebnisse im Cache erfährt man über die Methode `getJobCount()` und die durchschnittliche Suchzeit im globalen System der im Cache gespeicherten Suchanfragen über `getAverageSearchTime()`. Auch dieser Wert ist wiederum beeinflussbar, indem man mit der Methode `getSearchTimeHistoryLength()` die Menge der bei der aktuellen Berechnung verwendeten Suchergebnisse auslesen und mit `setSearchTimeHistoryLength()` einstellen kann. Nach einer Neujustierung kann die Berechnung der durchschnittlichen Suchzeit mit den neuen Parametern über die Methode `updateSearchTimeHistory()` neu ausgelöst werden.

Wie am Anfang dieses Abschnitts schon erwähnt, werden die nicht gecachten Suchaufträge wegen der nicht vorhersagbaren Suchzeit im globalen Netzwerk asynchron abgewickelt. Eine abgesetzte Suchanfrage muss daher von der suchenden Instanz regelmäßig auf Suchfortschritt überprüft werden, was ihr über die Benutzung der Methode `getJobStatus()` ermöglicht wird. Zur Aktualisierung des Suchanfragenstatus implementiert `StorageService` die Methoden des P-Grid-Interfaces *SearchListener* `searchStarted()` (für Suche gestartet), `searchedFailed()` (für Suche abgebrochen), `noResultFound()` (für kein Ergebnis gefunden), `newSearchResult()` (für weiteres Ergebnis gefunden) und am Ende einer jeden Suche `searchFinished()` (für Suche beendet), welche nach dem *Listener-Pattern* eingebunden werden und beim Eintreten des mit ihnen korrespondierenden Ereignisses den Status des Suchauftrages entsprechend setzen. Die möglichen Zustände eines Suchauftrages und das Verhalten des Caches wird detailliert in Abschnitt 9.7.5 beschrieben, in dem das Cache-Protokoll APLICOOVERs behandelt wird.

9.3.3 Die Klasse SearchJob

Im letzten Abschnitt wurde schon die Klasse `SearchJob` angesprochen, deren Objekte jeweils alle Daten und Zustände einer Suchanfrage verkörpern. Wird ein solches Objekt erzeugt, werden unter anderem auch

- die Startzeit der Suchanfrage,
- ein eindeutiger Identifizierer der Suchanfrage,
- der gesuchte Dienstname und
- sein Datentyp

erzeugt und gespeichert. Die gespeicherten Werte können jederzeit über die Methoden `getQueryGUID()`, `getQueryString()` und `getQueryStringType()` der `SearchJob`-Objekte abgerufen werden. Bei einem Aufruf der gecachten Suchmethoden von `StorageService` wird z.B. auf die Methoden `getQueryString()`

und `getQueryStringType()` der im Cache gespeicherten Suchobjekte zurückgegriffen, um eine geeignete Antwort auf eine neue Suchanfrage zu finden. Sie wird dann mittels der Methode *getSearchResults()* zurückgeliefert.

Weitere Methoden in `SearchJob` sind *setResults()* und *closeJob()*. Die letztere wird bei der Beendigung einer Suche im globalen Netzwerk aufgerufen, um die mit diesem Ereignis verbundene aktuelle Zeit zu speichern und die benötigte Suchzeit des Suchauftrages aus der Endzeit und der Startzeit zu berechnen. Mit den Methoden `getEndTime()` und `getSearchTime()` können diese Werte dann aus dem `SearchJob`-Objekt ausgelesen werden. Die restlichen Methoden *getStatus()* und *setStatus()* korrespondieren mit den Methoden *getJobStatus()* und *setJobStatus()* `StorageServices`. Sie lesen bzw. setzen den Status der Suchanfrage im jeweiligen `SearchJob`-Objekt im Suchanfragen-Cache `StorageServices`.

9.4 Die Softwarekomponente Statistics

Die Softwarekomponente `Statistics` bildet die dritte Komponente APLICOOVERs. Sie ist in der gleichnamigen Klasse als Thread implementiert und misst permanent in regelmäßigen Abständen Leistungsparameter von APLICOOVER und P-Grid. So werden u.a. die aktuellen Werte

- der verbrauchten Bandbreite im globalen Netzwerk,
- der Anzahl der Verwaltungs- und Suchnachrichten sowie
- Suchzeiten

gemessen. Die Leistungswerte können für die Bewertung des aktuellen Belastungszustandes des globalen Netzwerkes dienlich sein und möglicherweise in zukünftigen Entwicklungstufen QuickLinkNets zur intelligenten Steuerung der APLICOOVER-Schicht dienen. Für die weitere Beschreibung von APLICOOVERs Funktionalität in dieser Dissertation ist dieses Modul allerdings nicht weiter relevant. Eine ausführliche Beschreibung der Funktionsumfänge kann man der API-Dokumentation im Anhang A.4 dieser Arbeit entnehmen.

9.5 Die Softwarekomponente Messaging

`Messaging` ist die letzte wichtige Komponente APLICOOVERs. Sie bietet über zwei Methoden die Möglichkeit, Nachrichten über die Overlaystruktur P-Grids zu senden und zu empfangen. In der aktuell implementierten Version QuickLinkNets wird diese Funktionalität durch den Infrastrukturdienst selbst nicht genutzt, aber über das Interface *RemoteAdministration* zur Verfügung gestellt.

9.6 Die Interfaces

Über die Interfaces können andere Softwaremodule QuickLinkNets, andere Dienste und MAS auf die Funktionalitäten APLICOOVERs zugreifen. Dabei fungiert APLICOOVER als Lieferant von Anwendungsdienstinformationen über fremde Regionen. APLICOOVER stellt über seine Komponente Core die Interfaces

- *RemoteAdministration* und
- *RemoteServices*

bereit. Während *RemoteAdministration* zur Steuerung APLICOOVERs selbst von externen Komponenten aus dient, ermöglicht *RemoteServices* die Zugriffe auf die Funktionen, die APLICOOVER als Dienstleistung anderen Diensten und verteilten Anwendungen zur Verfügung stellt.

9.6.1 RemoteAdministration

Das Interface *RemoteAdministration* dient der Verwaltung APLICOOVERs und ist in der gegenwärtigen Implementierung auf die Erfordernisse ServiceJugglers zugeschnitten. Es könnte aber ebenso von einer anderen verwaltenden Softwarekomponente, einem Dienst oder einer verteilten Anwendung wie einem MAS bedient werden. *RemoteAdministration* bietet Methoden zum

- Starten und Initialisieren (`init()`) sowie
- Beenden (`shutdown()`) von APLICOOVER,
- Veröffentlichen des ganzen Regionsprofils im globalen Netzwerk (`publishRegionProfile()`),
- Löschen des ganzen Regionsprofils im globalen Netzwerk (`deleteRegionProfile()`),
- Setzen der Cachezeit von APLICOOVERs Cacheprotokoll (`setCacheTime()`),
- Auslesen der Cachezeit (`getCacheTime()`) und
- Routen von Nachrichten (`routeMessage()`) über die Infrastruktur des globalen Netzwerkes (Tunneling) an.

Zwischen den über die Methoden ausgelösten Vorgängen bestehen einige Abhängigkeiten. Das Starten und Initialisieren von APLICOOVER sollte die Veröffentlichung des *Regionsprofils* nach sich ziehen. Der Begriff Regionsprofil entspricht im Zusammenhang mit den Funktionen APLICOOVERs den zusammengefassten, komprimierten Anwendungsdienstinformationen einer Region, die von

ServiceJuggler erzeugt und an APLICOOVER weitergegeben werden. Bei Beendigung von APLICOOVER wird auch das Regionsprofil der Region gelöscht.

Auf das Thema Cachezeiten wird bei der Behandlung des Cacheprotokolls APLICOOVERs in Abschnitt 9.7.5 eingegangen. Die Methode *routeMessage()* ermöglicht das Versenden von Nachrichten verteilter Anwendungen oder Dienste über die Infrastruktur APLICOOVERs. Dies erfolgt über getunnelte Verbindungen zu anderen APLICOOVER-Knoten im globalen Netzwerk über die Overlaystruktur APLICOOVERs. In der aktuellen Implementierung QuickLinkNets wird diese Funktionalität aber von den anderen Infrastrukturdiensten ServiceJuggler und QuickLink nicht verwendet. Es stellt aber z.B. für verteilte Anwendungen eine Möglichkeit dar, mit entfernten Instanzen in anderen Regionen zu kommunizieren.

9.6.2 RemoteServices

Dieses Interface verkörpert die zweite Gruppe von APLICOOVERs Funktionalitäten, die mit seinen eigentlichen Dienstleitungen gegenüber verteilten Anwendungen assoziiert ist. Es bietet Methoden

- zur Veröffentlichung von Anwendungsdienstinformationen im globalen Netzwerk (`publishService()`),
- zum Aktualisieren von Anwendungsdienstinformationen (`updateService()`) und
- zum Löschen von Anwendungsdienstinformationen (`deleteService()`) an.

Da in QuickLinkNet die Steuerung der Veröffentlichung von Anwendungsdienstinformationen ServiceJuggler übernimmt, dürfen die entsprechenden Methoden zum Einstellen und Löschen von Dienstinformationen nur von diesem bedient werden. Soll APLICOOVER unabhängig von QuickLinkNet eingesetzt werden, können diese Methoden von allen verteilten Anwendungen und Diensten benutzt werden, die Dienste global propagieren wollen.

Zum Suchen nach Anwendungsdienstinformationen bietet RemoteServices Methoden

- zur Suche nach Anwendungsdiensten im globalen Netzwerk (`query()`),
- zur Suche nach Anwendungsdiensten im globalen Netzwerk auf Basis eines Teilstrings des Dienstnamens (`rangeQuery()`),
- zur Suche nach Anwendungsdiensten im globalen Netzwerk mit vorheriger Suche in APLICOOVERs Cache (`cachedQuery()`) und

- zur Suche nach Anwendungsdiensten im globalen Netzwerk mit vorheriger Suche in APLICOOVERs Cache auf Basis eines Teilstrings des Dienstnamens
  (`cachedRangeQuery()`).

Da die Aufrufe zur Suche nach Anwendungsdienstinformationen asynchron erfolgen, braucht man für eine Anfrage auch zusätzliche Informationen, um die Asynchronität handhaben zu können. RemoteServices bietet deshalb Methoden

- zur Ermittlung der aktuellen durchschnittlichen Suchdauer (`getAverageSearchTime()`),
- zum Abfragen des Status des Suchaufrufes (`getQueryStatus()`) und
- zum Abfragen der Suchergebnisse (`getSearchResults()`) an.

9.7 Funktionsweise und Vorgänge APLICOOVERs

Nachdem die Aufgaben und der Aufbau der einzelnen Komponenten APLICOOVERs beschrieben wurden, wenden sich die folgenden Abschnitte den wesentlichen Vorgängen in APLICOOVER zu, die durch das Zusammenspiel der Komponenten geleistet wird. Dabei wird auch auf das Zusammenspiel mit ServiceJuggler und den anderen beteiligten Diensten und Anwendungen eingegangen.

9.7.1 Start, Bootstrapping und Herunterfahren

Der Start APLICOOVERs kann in die zwei Phasen *Start und Initialisierung* und *Bootstrapping* eingeteilt werden.

Die erste Phase wird von ServiceJuggler ausgelöst. Zunächst wird ein Objekt der Klasse `Core` und der P-Grid Klasse `P2PFactory` erzeugt. Anschließend wird das `Core`-Objekt an der RMI-Registry angemeldet. `Core` als zentraler Baustein APLICOOVERs holt sich nun die Referenz der einzigen Instanz von P-Grids `P2PFactory`-Klasse. Nun übergibt `Core` der `P2PFactory` alle Parameter zum Start einer Peer-to-Peer-Instanz, worauf die `P2PFactory` ein entsprechend initialisiertes Objekt `P2PFacility` erzeugt. Dieses stellt nun das zentrale Objekt der lokalen P-Grid-Instanz und die zentrale Verbindung von APLICOOVER zum globalen P-Grid Peer-to-Peer-Netzwerk dar.

Im nächsten Schritt erzeugt `Core` ein `StorageService`-Objekt und übergibt ihm die Referenz auf die `P2PFacility`, so dass `StorageService` Zugriff auf P-Grids Storagefunktionalität bekommt. `Core` schließt diesen Schritt ab, indem es das `StorageService`-Objekt mit der Beschreibung des im Speichermodul zu verwendenden Datentyps ("service-description") versieht, damit dieses die später im Betrieb übergebenen Informationen richtig handhaben kann. Nach diesem Schritt ist die grundsätzliche Funktionalität APLICOOVERs hergestellt

und die Verbindung zum globalen Netzwerk könnte aufgenommen werden. Bevor dies aber geschieht, wird APLICOOVER aber noch weitere Funktionalität hinzugefügt. Dazu gehören die Services `Statistics`, welches dem Monitoring der Vorgänge im System dient, und `MessageService` zur Kommunikation mit anderen APLICOOVER/P-Grid Peers im globalen Netzwerk.

Nun beginnt die zweite Phase, das *Bootstrapping*, d.h. die Kontaktaufnahme und der Zutritt(sversuch) zum globalen Netzwerk. Dazu speichert APLICOOVER eine Liste von Peers, mit denen es zuvor schon verbunden war. Die Liste wird nun nacheinander abgearbeitet, d.h. es wird versucht, jeweils eine Verbindung zu diesen Peers aufzunehmen. Die Versuche der Verbindungsaufnahmen selbst erfolgen asynchron, da die Antwortzeiten von Peers im globalen Netzwerk recht groß sein können.

Wird APLICOOVER zum ersten Mal gestartet, so existiert natürlich keine Liste von ehemals verbundenen Peers. In diesem Fall muss auf die IP-Adresse eines Default-Bootstrap-Peers zurückgegriffen werden, welche beim Initialisieren von `Core` mit übergeben wird. Kann dieser Default-Bootstrap-Peer ebenfalls nicht erreicht werden, gründet APLICOOVER sein eigenes Netzwerk mit sich selbst als einzigen Peer und wartet auf Verbindungen von anderen Peers. Dieser Zeitpunkt stellt das Ende des Bootstrapping-Vorgangs dar, an dem der aufrufenden Instanz, also ServiceJuggler, ein Flag mit dem entsprechenden Status des Verlaufs des Bootstrapping zurückgegeben wird. APLICOOVER ist nun vollständig einsatzbereit.

Das Herunterfahren APLICOOVERs folgt im Prinzip der umgekehrten Reihenfolge wie Bootstrapping und Initialisierung. Alle verbundenen Module werden zum Speichern ihrer Daten aufgefordert und beenden sich dann ihrerseits. Dabei werden auch die Verbindungen zu P-Grid gekappt und die entsprechenden Objekte an der RMI-Registry abgemeldet.

9.7.2 Ein Regionsprofil veröffentlichen, aktualisieren und löschen

Im normalen Ablauf folgt nach dem Start und dem erfolgreichen Bootstapping die Veröffentlichung des Regionsprofils im globalen Netzwerk. Dieser Vorgang wird ebenfalls durch ServiceJuggler ausgelöst. Zuvor muss ServiceJuggler aus den Einträgen seines ServiceRegistry das Regionsprofil erstellt haben.

Ein Regionsprofil stellt eine reduzierte Darstellung der Anwendungsdienste einer Region dar. Zur Reduzierung der Anwendungsdienstinformationen wird nicht jeder Dienst einzeln genannt, sondern alle gleichen und stabilen Dienste auf verschiedenen Plattformen in einer Region werden auf eine Dienstbeschreibung reduziert und mit der Anzahl ihres Vorkommens in der Region sowie der IP-Adresse des Regions-Managers versehen. Der Eintrag eines Anwendungsdienstes im Regionsprofil hat folgende formale Form:

```
Dienstbeschreibung := Dienstname + '#' + Kardinalität .
```

Die in einem Eintrag des Regionsprofils enthaltene Information ist für einen MA in einer entfernten Region ausreichend, um eine Route zu planen und zu entscheiden, ob die betrachtete Region den gewünschten Anwendungsdienst anbietet und ob er diese besuchen will oder nicht. Beim Vorliegen mehrerer Regionen, welche den gewünschten Dienst anbieten, hilft die Angabe der Kardinalität des Dienstes in der Region, um eine quantitativ gestützte Auswahl zu treffen. So ist eine Region, die den gewünschten Dienst mehrfach anbietet, für einen MA die bessere Wahl für seine Routenplanung, da selbst beim Ausfall eines den Dienst anbietenden Knotens in der Zielregion dort noch andere Plattformen den Dienst anbieten.

Zur Veröffentlichung des Regionsprofils gibt ServiceJuggler dessen Diensteinträge sequentiell an APLICOOVER weiter, in welchem sich das Storage-Modul um die Umwandlung der Dienstbeschreibung in eine für P-Grid verarbeitbare Form und die Weitergabe an dieses kümmert. Der weitere Veröffentlichungsprozeß im P-Grid-Netzwerk geschieht dann asynchron, so dass eine schnelle Übergabe der einzelnen Diensteinträge des Regionsprofils von ServiceJuggler über APLICOOVER an die P-Grid-Schicht sichergestellt ist.

Die P-Grid-Schicht gibt ihrerseits für jeden eingefügten Diensteintrag einen global eindeutigen Identifizierer (*GUID*) zurück, welche in APLICOOVER gespeichert wird. Diese Identifizierer werden benötigt, um bei Aktualisierungen oder beim Entfernen der Diensteinträge diese im globalen Netzwerk eindeutig zuordnen zu können. Die GUIDs haben folgende formale Form:

```
Veröffentlichter_Dienst := Dienstname + '_~_' + Dienst_GUID .
```

Natürlich bietet APLICOOVER auch eine Funktion für ServiceJuggler an, um das ganze Regionsprofil auf einmal zu löschen. Wird die entsprechende Methode angesprochen, löschen APLICOOVER und P-Grid die Diensteinträge der Region, analog zum Veröffentlichen der Diensteinträge im globalen Netzwerk, sequentiell, d.h. Diensteintrag für Diensteintrag.

9.7.3 Die Suche nach Anwendungsdiensten im globalen Netzwerk

Wie in Abschnitt 9.3.2 vorgestellt, kann man in APLICOOVER nach zwei Arten suchen: Im lokalen Cache oder im globalen Netzwerk. Dieser Abschnitt widmet sich der Suche im globalen Netzwerk, die mit der Methode `query()` ausgelöst wird und in APLICOOVER in drei Schritten vonstatten geht: Im ersten Schritt wird die Suchanfrage entgegen genommen, für P-Grid aufbereitet und anschließend an dieses weitergegeben. Im zweiten Schritt erfolgt die eigentliche Suche im globalen P-Grid-Netzwerk, und die Ergebnisse werden empfangen. Im Schritt

drei wird das Ergebnis von APLICOOVER verarbeitet, interne Wartungsarbeiten ausgeführt und die suchende Instanz, welche die Suche ausgelöst hat, vom Ergebnis der Suche benachrichtigt.

9.7.3.1 Die Aufbereitung der Suchanfrage

Die Suche beginnt mit einer Suchanfrage an APLICOOVERs `Core`, bei dem der Name des gesuchten Anwendungsdienstes als Parameter in Form eines Strings übergeben wird. `Core` bindet den Dienstnamen, wie in Abschnitt 9.3.1 beschrieben, an den Datentypen `description`, da für eine Suche im P-Grid-Netzwerk ein Datentyp zwingend notwendig ist. Nun wird die aufbereitete Suchanfrage an `Storage` weitergegeben. Dieses Modul ruft nun seinerseits die Methode `createQuery()` der P-Grid-Schicht auf, wobei die aufbereitete Suchanfrage als Parameter übergeben wird. Die P-Grid-Schicht generiert daraus ein Suchobjekt, versieht es mit einem eindeutigen Identifizierer (*GUID*) und liefert es an `Storage` zurück. Im nächsten Schritt generiert `Storage` ein zum Suchobjekt passendes `SearchJob`-Objekt und speichert es im Cache. Dieses zur Suchanfrage gehörende `SearchJob`-Objekt speichert alle zukünftig erhaltenen Antworten und den aktuellen Status der Suchanfrage. Die GUID wird dabei zur eindeutigen Zuordnung von Antworten des P-Grid-Suchobjektes zum entsprechenden `SearchJob`-Objekt verwendet.

9.7.3.2 Der Ablauf der Suche

Der zweite Schritt der Suche beginnt mit der Übergabe des zuvor von P-Grid erzeugten und mit seinem `SearchJob`-Objekt assoziierten Suchobjektes an die P-Grid-Schicht. Dadurch wird die Suche im globalen Netzwerk ausgelöst. Die Suchanfrage wird an einen der in der Suchstruktur des P-Grid-Netzwerkes für die Suchanfrage zuständigen Peers geroutet, der die Suchanfrage mit einem oder mehreren Ergebnissen beantwortet. Die P-Grid-Schicht aktualisiert mit jedem eintreffenden Ergebnis simultan das `SearchJob`-Objekt, so dass dieses immer den aktuellen Status und die Ergebnisse der Suchanfrage widerspiegelt. Die suchende Instanz kann sich über den Aufruf der Methode `getQueryStatus()` in Verbindung mit der GUID so jederzeit über den Status seiner Suchanfrage informieren und Zwischenergebnisse abholen, falls schon Ergebnisse vorliegen und die Suche noch nicht beendet ist. Ist die Suche im globalen Netzwerk beendet, setzt P-Grid im `SearchJob`-Objekt den entsprechenden Status und beendet den Suchvorgang APLICOOVERs.

9.7.3.3 Die Nachbereitung und Verarbeitung der Suchergebnisse

Wird eine Suche von P-Grid beendet, führt `Storage` noch einige Arbeiten aus, um die Suche seinerseits zu beenden und den Cache zu warten. Zuerst wird der Endzeitpunkt der Suche festgehalten, und die statistischen Werte der in Abschnitt 9.4 aufgeführten Zeiten werden berechnet bzw. aktualisiert. Ferner wird der Cache gewartet, dessen Funktionsweise in Abschnitt 9.7.5 genauer beschrieben wird. Als letztes wird die suchende Instanz, also z.B. ein MA, über ihre beendete Suchanfrage benachrichtigt. Die suchende Instanz konnte, nachdem sie die Suchanfrage absetzte, in der Zwischenzeit weiterarbeiten, während die Suchanfrage selbst asynchron verarbeitet wurde. Mit der Benachrichtigung wird die suchende Instanz über den Abschluss der Suche informiert und ihr die GUID des zu ihrer Anfrage gehörenden `SearchJob`-Objektes zurückgegeben. Sie kann nun, wie und wann sie möchte, die vollständigen Ergebnisse mit Hilfe der GUID über die Methode `getSearchResults()` abholen. Die einzelnen Ergebnisse (Regionseinträge) haben das Format

```
Gefundener_Dienst := Dienstname + '#' + Kardinalität + '@' +

                     IP-Adresse
```

und werden als *String* zurückgeliefert. In ihm sind jeweils der Dienstname, die Anzahl des Vorkommens des Dienstes in der entfernten Region und die IP-Adresse des entfernten Regions-Managers gespeichert.

9.7.4 Andere Suchmethoden in APLICOOVER

APLICOOVER unterstützt, wie in den Abschnitten 9.3.2 und 9.6 beschrieben, mehrere Arten der Suche nach Anwendungsdiensten. So kann die Inanspruchname einer Bereichsuche, d.h. nach einem Teilstring des Dienstnamens, im globalen Netzwerk über die Methode `rangeSearch()` ausgelöst werden. Die Suche erfolgt dann analog zu der in Abschnitt 9.7.3.2 beschriebenen. Die gecachten Methoden `cachedSearch()` und `cachedRangeSearch()` durchsuchen im Gegensatz dazu jeweils nur den lokalen Cache nach erfolgreich abgeschlossenen `SearchJobs`, die mit der Suchanfrage matchen. Diese beiden Methoden können über ein Flag, welches der Suchanfrage als Parameter mit übergeben wird, dazu veranlasst werden, bei einer nicht erfolgreichen Suche im lokalen Cache automatisch eine globale Suche, wie sie in Abschnitt 9.7.3 beschrieben ist, auszulösen.

9.7.5 Das Cacheprotokoll APLICOOVERs

Der Cache APLICOOVERs stellt einen Container zum Speichern von `SearchJob`-Objekten dar. Er speichert dabei gleichzeitig die Informationen über entfernte

Regionen, die der eigenen bekannt sind. Diese bekannten Regionen führen im globalen P-Grid-Overlaynetzwerk zu den erwünschten Shortcuts, welche das *Small-World-Phänomen* in APLICOOVER erzeugen.

Der Cache wird für die Suche von entfernten Anwendungsdiensten in einer lokalen Datenbasis verwendet, was die Suche enorm beschleunigen kann. Da die Einträge im Cache mit zunehmendem Alter an Zuverlässigkeit verlieren, müssen sie gewartet werden. Dies erfolgt mit Hilfe eines Cacheprotokolls, welches die `SearchJob`-Objekte mit verschiedenen Status, entsprechend ihres inneren Zustandes, versieht. Abbildung 9.3 zeigt die möglichen Zustände eines `SearchJob`-Objektes und ihre Abhängigkeiten und Übergänge untereinander.

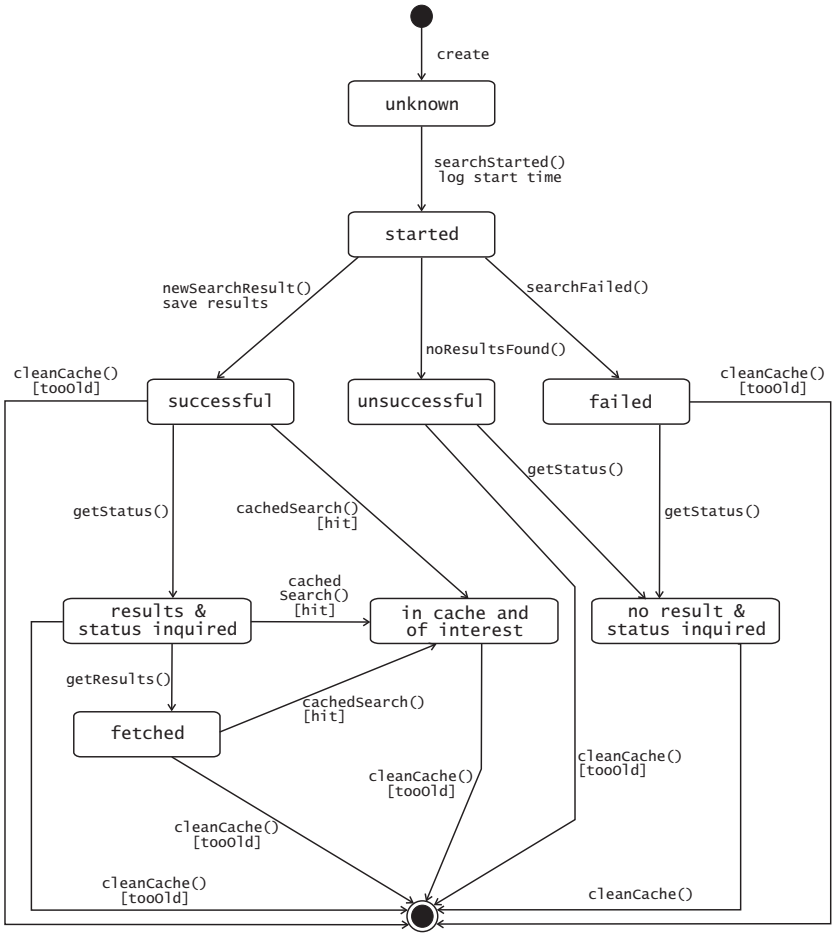

Abbildung 9.3: Das Cache-Protokoll APLICOOVERs, in Anlehnung an [Hen06]

Wenn ein neues `SearchJob`-Objekt angelegt wird, ist sein Status erst einmal *unbekannt (unknown)*. Diesen Zustand behält es bei, bis die Suchanfrage an die P-Grid-Schicht übergeben und die Suchanfrage ausgelöst wird. Die Vorgänge,

die sich zwischen APLICOOVER und P-Grid in diesem Zustand abspielen, sind in Abschnitt 9.7.3.1 beschrieben.

Mit der Übergabe einer Suchanfrage an P-Grid durch den Aufruf der Methode `searchStarted()` und deren Start ins globale P-Grid-Netzwerk wird die Startzeit der Suche genommen und der Status des `SearchJob`-Objektes auf *gestartet (started)* gesetzt. Ab diesem Zeitpunkt wird die Steuerung von APLICOOVER an P-Grid abgegeben. Das Objekt bleibt in diesem Zustand, bis ein Ergebnis aus dem globalen P-Grid-Netzwerk eingetroffen und die Steuerung an APLICOOVER zurückgegeben ist.

Das Ergebnis der bis hierher asynchon ablaufenden Suche kann nun entweder *erfolgreich (successful)*, wenn ein Ergebnis gefunden wurde, *nicht erfolgreich (unsuccessful)*, wenn kein Ergebnis gefunden wurde, oder *fehlerhaft (failed)*, wenn ein technischer Fehler bei der Suche auftrat, sein. Der Status des `SearchJob`-Objekts wird dann jeweils in den entsprechenden Folgezustand überführt, von dem aus sich die suchende Instanz mit der Methode `getStatus()` über das Ergebnis der Suche informieren kann[8]. Der Aufruf dieser Methode führt das `SearchJob`-Objekt, abhängig von seinem konkreten Zustand und seiner Situation, in den jeweils nächsten Zustand über.

Wurde die Suchanfrage erfolgreich beendet, geht der Zustand des Objektes in *result and status inquired* über. Die gecachten Suchmethoden, die nur den Cache selbst durchsucht haben, überführen ihr `SearchJob`-Objekt aus diesem oder direkt aus dem vorherigen Zustand in den Zustand *in cache and of interest* über. Dieser Zustand spiegelt ein erfolgreiches und weiterhin benötigtes Suchergebnis wider. Eine ungecachte Suchmethode wartet im Zustand *result and status inquired*, bis die suchende Instanz die Ergebnisse mittels der Methode `getResult()` abgeholt hat und geht dann in den Zustand *fetched* über.

Wurde die Suchanfrage hingegen nicht erfolgreich *(unsuccessful)* beendet oder verlief die Suche fehlerhaft *(failed)*, ist kein Ergebnis vorhanden. Sobald die suchende Instanz den Status einer solchen Suchanfrage abgerufen hat, geht das `SearchJob`-Objekt in den Status *no result and status inquired* über, der anzeigt, dass das Objekt respektive sein Suchergebnis nicht mehr gebraucht wird. Beim nächsten Aufruf der Methode cleanCache() wird das `SearchJob`-Objekt dann sofort gelöscht.

Von allen Zuständen außer *unknown* und *started* aus wird ein `SearchJob`-Objekt aus dem Cache entfernt, wenn es die maximale Zeit des Verbleibs im Cache erreicht hat. Damit werden zu alte oder verwaiste Sucheinträge gelöscht und die gecachten Daten auf einem Mindeststand in Bezug auf Aktualität gehalten.

[8] Die Benachrichtigung der suchenden Instanz und deren Möglichkeiten, sich die Ergebnisse abzuholen, sind in Abschnitt 9.7.3.3 auf Seite 181 beschrieben. Aufgrund der asynchronen Suche in APLICOOVER muss die suchende Instanz das Ergebnis nicht zwingend abholen.

9.8 Zusammenfassung

Der Infrastrukturdienst APLICOOVER bildet die oberste Schicht in QuickLink-Net und baut ein globales logisches Netzwerk nach einer Peer-to-Peer-Architektur auf. Dieses globale Netzwerk verbindet Regionen von QuickLink-Plattformen miteinander und erfüllt die Anforderungen an eine globale Suchstruktur. Als unterliegende Technologie wird P-Grid verwendet, welches Eigenschaften wie eine globale Skalierbarkeit und eine hinreichend gute Unterstützung von Netzwerkdynamik garantiert.

Im von APLICOOVER gebildeten globalen P-Grid-Netzwerk werden reduzierte Anwendungsdienstinformationen ganzer Regionen veröffentlicht, nach denen mit verschiedenen Methoden gesucht werden kann. Dadurch wird die abzubildende Dynamik und die Anforderungen an die Skalierbarkeit im globalen System QuickLinkNets zusätzlich verringert, was sich leistungssteigernd auf das Gesamtsystem auswirken kann.

Zusätzlich stellt ein lokaler Cache von Suchergebnissen in jeder APLICOOVER-Instanz einerseits eine gute Suchperformance bei oft nachgefragten Anfragen sicher und sorgt andererseits gleichzeitig für die Ausbildung von Shortcuts zu anderen interessanten Regionen im globalen Netzwerk. Diese zusätzlich zum binärbaumbasierten P-Grid-Overlay entstehende, interessenbetonte Verlinkung bewirkt eine als *Small-World-Phänomen* bekannte Clusterung, die den ohnehin schon guten Suchaufwand von binärbaumbasierten Systemen nochmals erheblich mindern kann.

Nach Diensten suchende Instanzen verteilter Systeme wie MA können in APLICOOVER Dienste suchen und bekommen bei erfolgreicher Suche die Ortsinformation mindestens einer Region (ihres Regions-Managers), in der der gesuchte Dienst vorkommt. Ein MA erfährt nach erfolgreicher Migration durch das globale Internet zum Regions-Manager mindestens einen konkreten Aufenthaltsort und Dienstzugangspunkt zum gesuchten Dienst in der neuen Region.

10 Zusammenfassung QuickLinkNet

QuickLinkNet ist ein komplexer Infrastrukturdienst, der in seiner Implementierung der in Kapitel 5 vorgeschlagenen Architektur folgt. Er löst das Problem der Bereitstellung aktueller Anwendungsdienstinformationen für mobile Agenten der Stufe 2 unter den mengenmäßigen Skalierungsbedingungen des globalen Internets. Die Verwaltung der Dienstinformationen wird mit einem transparenten zweistufigen Ansatz gelöst. Die zwei Stufen des Ansatzes spiegeln sich in mehreren Gesichtspunkten QuickLinkNets wider. So werden Dienste innerhalb QuickLinkNets in zwei abstrakte Ebenen aufgeteilt und getrennt verwaltet:

Grundsätzliche Dienste mit technischer Natur, die die Problemlösungsmöglichkeiten der Anwendungen verbessern, also WIE eine Aufgabe gelöst werden kann, bestimmen die untere Ebene. Dazu gehören auch die Infrastrukturdienste, wie QuickLinkNet und seine Komponenten sie darstellen. Diese Dienste spielen für verteilte Anwendungen wie MA der Stufe 2 nur in ihrer unmittelbaren Netzwerkumgebung eine Rolle und sind für eine schnelle Orientierung von MA in lokalen logischen Netzwerken nötig. Die Dienstinformationen dieser Dienste unterliegen daher einer hohen Anforderung an die Aktualität. Diese Art der Dienste wird in QuickLinkNet durch die Komponente QuickLink verwaltet. QuickLink unterstützt Aktualisierungsraten im Sekundenbereich.

Anwendungsdienste bestimmen die obere abstrakte Ebene der Dienste. Sie können erst durch die Inanspruchnahme von (Infrastruktur-)Diensten der unteren Ebene detektiert werden, wie zum Beispiel einem Verzeichnisdienst für Anwendungsdienste (ServiceRegistry). Anhand dieser Dienste routet sich ein MA der Stufe 2 durch das Internet, um seine Hauptaufgabe, die in seinem User-Task gekapselt ist, erfüllen zu können. Für die Anwendungsdienste benötigt der MA eine inhaltliche Beschreibung, also WAS ein Dienst leisten kann, und eine Ortsinformation, also WO der Dienst in Anspruch genommen werden kann. Die Dienstleistung, diese Informationen MA zur Verfügung zu stellen, wird in QuickLinkNet durch die Komponenten ServiceJuggler und APLICOOVER geleistet.

Ein zweistufiger Ansatz wird wiederum auch bei der Verwaltung der Anwendungsdienstinformationen verwendet. Bezüglich dieser wird hier in eine lokale und eine globale Sicht auf das Netzwerk unterschieden, die sich auch technisch

in einem lokalen System, gebildet durch QuickLink und ServiceJuggler, und einem globalen System, gebildet durch APLICOOVER, widerspiegeln:

Die lokale Sicht betrachtet nur die Anwendungsdienste in der unmittelbaren Umgebung des mobilen Agenten, der Region. Sie betont dabei bei mäßiger Mengenskalierbarkeit vor allem die Vollständigkeit der Anwendungsdienstinformationen in Verbindung mit einer relativ hohen abbildbaren Dynamik.

Die globale Sicht auf das Netzwerk dient dagegen der groben Orientierung eines MA über große Entfernungen hinweg. Deshalb wird hier vor allem auf die Mengenskalierbarkeit Wert gelegt, während die abbildbare Dynamik deutlich niedriger und die Detailtiefe der Dienstinformation geringer gegenüber der lokalen Sicht ist.

Das gesamte System QuickLinkNet ermöglicht die effektive und effiziente Versorgung der durch Erfurth [Erf04] eingeführten Routingdienste mit den notwendigen aktuellen Anwendungsdienstinformationen, durch die ein autonomes Routing von MA der Stufe 2 erst möglich wird. Ebenfalls unterstützt QuickLinkNet durch seine getrennten Sichtweisen auf das Netzwerk die durch Erfurth eingeführten Fish-Eye-View-Karten [ER02] für MA der Stufe 2, welche die nähere Umgebung des MA scharf und detailliert, die entfernte Umgebung dagegen undeutlicher, also weniger detailreich, darstellt.

Obwohl QuickLinkNet speziell zur Unterstützung mobiler Agenten der Stufe 2 implementiert wurde, unterstützt es ebenso andere verteilte Anwendungen, welche auf der Nutzung von entfernten Diensten beruhen.

Komponente aktiviertes Modul	Rolle des QuickLink-Knotens		
	minimaler Knoten	normaler Knoten	Regions-Manager
QuickLink			
alle Module, außer PerformanceAnalyser	x	x	x
PerformanceAnalyser	o	o	o
ServiceJuggler			
ServiceRegistration	o	x	x
ServiceRegistry	-	-	x
ServiceManager	-	x	x
APLICOOVER	-	-	x

Tabelle 10.1: QuickLink-Knoten in verschiedenen Situationen und Rollen und die aktivierten, dafür benötigten Komponenten QuickLinkNets

Den Zusammenhang zwischen den verschiedenen Rollen der Knoten eines logischen QuickLinkNet-Netzwerkes und der dazu aktivierten Infrastrukturdienst-

komponenten des Infrastrukturdienst-Frameworks QuickLinkNet verdeutlicht Tabelle 10.1. Die aktiv benötigten Module und Komponenten sind dort mit einem 'x', die nicht benötigten mit einem '-' und die optionalen mit einem 'o' gekennzeichnet.

III

Evaluierung QuickLinkNet

11 QuickLink

Diese Kapitel befasst sich mit der Evaluierung der prototypischen Implementierung QuickLinks, welches innerhalb QuickLinkNets die lokale Sicht auf das Netzwerk abbildet und einen lokalen Netzwerkbereich verwaltet. Alle Komponenten QuickLinks sind praktisch getestet und bezüglich der gewünschten Performanceansprüche evaluiert worden und zeigen, dass diese auch erfüllt werden können.

QuickLinks Aufgaben lassen sich in zwei Bereiche einteilen:

1. Vernetzung und Lieferung grundsätzlicher Information über andere Plattformen im lokalen Netzwerk und
2. die Ermittlung von Leistungsparametern der Plattformen.

Den ersten Teil diese Aufgabe löst QuickLink rein broadcastbasiert, um den besonderen leistungsbezogenen Randbedingungen, die an diesen Infrastrukturdienst gestellt werden, gerecht zu werden. Die daraus folgenden netzwerkbezogenen Systemeigenschaften werden in den Abschnitten 11.1 bis 11.3 dargestellt.

Der zweite Teil der Evaluation QuickLinks in Abschnitt 11.4 beschäftigt sich mit der Ermittlung der leistungsbezogenen Werte der QuickLink-Plattformen, die für die Steuerung einer Region durch ServiceJuggler verwendet werden. Abschnitt 11.5 zieht ein Fazit aus der Evaluierung QuickLinks und schließt dieses Kapitel ab.

11.1 Randbedingungen der Vernetzung

QuickLink vernetzt die QuickLink-Plattformen im Bereich eines lokalen IP-Subnetzwerkes und führt damit zu einem logischen lokalen Netzwerk, welches ServiceJuggler auf einer höheren abstrakten Ebene zu einer Region zusammenführt. Eine grundsätzliche Bedingung beim Entwurf QuickLinks war dabei die Berücksichtigung hochdynamischer Agentenplattformen, die schnellstmöglich eingebunden und mit den notwendigen Dienstinformationen versorgt werden sollen, um ihren MA eine schnelle Migration auf andere, stabilere Plattformen ermöglichen zu können. Zusätzlich sollten die auf der Ebene von QuickLink bereitgestellten Dienstinformationen ein Höchstmaß an Aktualität besitzen, die Netzwerkbelastung aufgrund der dazu nötigen auszutauschenden Verwaltungsnachrichten und die in Anspruch genommene Rechenleistung der Hosts aber gering sein.

QuickLinks Kommunikation beruht daher auf Broadcastnachrichten, welche im Internetprotokollstapel mit Hilfe des unzuverlässigen Dienstes UDP realisiert werden.

Die Evaluierung QuickLinks beleuchtet sein Verhalten bezüglich dieser Eigenschaften näher. Es wurden:

- die Netzwerkbelastung,
- Zuverlässigkeit der Netzwerkkommunikation,
- die Eintrittsgeschwindigkeit in das logische Netzwerk und
- die benötigte Rechenzeit zur Verarbeitung der Netzwerknachrichten

betrachtet und untersucht. Die folgenden Abschnitte behandeln die aufgezählten Themenbereiche.

11.2 Netzwerkbelastung durch QuickLink

Die durch ein verteiltes System selbst hervorgerufene Netzwerklast berechnet sich aus der Summe der Nachrichten in einer bestimmten Zeit, die es zur Selbstverwaltung versendet. In QuickLink sind dies die

- Zyklusnachrichten (cycle messages),
- Beitrittsnachrichten (join messages) und
- Aktualisierungsnachrichten (update messages).

Beitritts- und Aktualisierungsnachrichten haben dabei den gleichen Aufbau und die gleiche Größe. Sie unterscheiden sich lediglich in ihrer Rolle, d.h. nach dem Grund, aus welchem sie gesendet werden. Für die Betrachtung der Paketgrößen spielt dies aber keine Rolle. Sie werden daher auch gemeinsam unter dem Begriff *Updatenachricht* zusammengefasst.

11.2.1 Paketgrößen

Die Nachrichten QuickLinks hängen, wie in Abschnitt 7.3.1.4 beschrieben, von den inneren Datenstrukturen QuickLinks ab. Die Nachrichtengrößen bestimmen wiederum direkt den Payload der Pakete der unterliegenden Netzwerkschichten. Daher werden nun zuerst die Größen der Nachrichten und die resultierenden Paketgrößen betrachtet, um danach mit Hilfe ihrer Sendefrequenzen auf die entstehenden Netzwerklasten schließen zu können.

11.2.1.1 Größe der Zyklusnachrichten

Die Nachrichtengröße $N_{maintenance}$ für eine Zyklusnachricht setzt sich aus der Headerlänge l_{header}, der IP-Adressenlänge $l_{address}$, der Dienstfeldlänge $l_{services}$ und der maximalen verwaltbaren Anzahl von QuickLink-Knoten pro logischem lokalen Netzwerk max aus

$$N_{maintenance} = l_{header} + (max * (l_{address} + l_{services}))$$

zusammen. Für die realistische Annahme

- einer Headerlänge l_{header} von 5 Byte,
- einer Adressenlänge $l_{address}$ von 4 Byte für eine IPv4-Adresse,
- einer Anzahl von 32 abbildbaren Diensten, die ebenfalls eine Dienstfeldlänge $l_{services}$ von 4 Byte ergeben und
- einer realistischen maximalen Anzahl max von 127 QuickLink-Knoten

berechnet sich die Nachrichtengröße $N_{maintenance}$ wie folgt:

$$N_{maintenance} = (5 + (127 * (4 + 4))) \, Byte = 1021 \, Byte.$$

Obwohl die hier als Beispiel herangezogene Zyklusnachricht im Vergleich zu einer Beitritts- oder Aktualisierungsnachricht recht groß erscheint, stellt sie absolut gesehen keine problematische Größe für ein modernes Netzwerk dar, wenn die Sendefrequenz gering bleibt.

11.2.1.2 Größe der Beitritts- und Aktualisierungsnachrichten

Die Nachrichtengröße $N_{dynamic}$ für eine Beitritts- oder Aktualisierungsnachricht setzt sich aus der Headerlänge l_{header}, der IP-Adressenlänge $l_{address}$ und der Dienstfeldlänge $l_{services}$ wie folgt zusammen:

$$N_{dynamic} = l_{header} + l_{address} + l_{services}.$$

Unter der realistischen Annahme

- einer Headerlänge l_{header} von 5 Byte,
- einer Adressenlänge $l_{address}$ von 4 Byte für eine IPv4-Adresse und
- einer Anzahl von 32 abbildbaren Diensten, die ebenfalls eine Dienstfeldlänge $l_{services}$ von 4 Byte ergeben,

berechnet sich die Nachrichtengröße $N_{dynamic}$ wie folgt:

$$N_{dynamic} = 5 + 4 + 4 \, Byte = 13 \, Byte.$$

Die reine Größe einer Beitritts- oder Aktualisierungsnachricht erscheint sehr klein.

11.2.1.3 Theoretisch und praktisch mögliche Nachrichten- und Paketgrößen

Java beschränkt die Menge der Daten beim Senden nicht, so dass man als Programmierer beim Senden von Daten zunächst alle Freiheiten bezüglich der Datenmenge besitzt. Berücksichtigt man aber die unterliegenden Netzwerkprotokolle, so können aus deren Eigenschaften unter Umständen Restriktionen bei der Bildung von Paketgrößen entstehen. Durch diese Restriktionen kann sich der Programmierer gezwungen sehen, die maximal versendbare Menge an Daten doch zu begrenzen. Im Falle von QuickLink ist dies so.

Im aktuellem Stand seiner Implementierung verwendet QuickLink zur Kommunikation ausschließlich Broadcastnachrichten, welche in einem IP-basierten Netzwerk über das unzuverlässige, verbindungslose Transportprotokoll User Datagram Protocol (UDP)[1][Pos80] abgewickelt werden. Um die Möglichkeit der fehlerhaften Übertragung fragmentierter QuickLink-Nachrichten zu vermeiden und damit die mögliche Gesamtfehlerrate von vornherein gering zu halten, sollten alle QuickLink-Nachrichten in jeweils einem einzigen Paket[2] übertragen werden. Dabei wird die maximale Größe eines solchen Pakets von allen verwendeten Protokollen des Protokollstapels bestimmt.

Während das Protokoll UDP die Datenmenge eines Paketes nach oben nicht begrenzt, tut dies das darunterliegende Internet-Protokoll sehr wohl, und zwar auf maximal 65536 Byte [Pos81a]. Da nur ein Paket verwendet werden soll, liegt die Grenze also erst einmal bei dem vom Internet-Protokoll bestimmten Wert.

Die unterhalb des Internetprotokolls eingesetzten Netzwerke können in Java nicht automatisch detektiert werden. Realistisch ist aber die Annahme, dass in typischen lokalen Netzwerkumgebungen am aller häufigsten die Protokolle Ethernet nach dem IEEE 802.3 [GLD+05] Standard und Wireless LAN nach IEEE 802.11x [ANS07] eingesetzt werden. Ein Fast-Ethernet nach IEEE 802.3 mit einer Datenrate von 100 MBit/s verwendet eine minimale Paketgröße von insgesamt 64 Byte, einen Header der maximalen Länge $l_{Eth-header}$ von 22 Byte und einen Trailer $l_{Eth-trailer}$ von 4 Byte. Das Datenfeld ist in seiner Größe variabel bis auf ein volles Byte, die Restbits werden mit sogenannten "Pads" gefüllt. Der maximale, im Zusammenspiel mit IP garantierte Payload beträgt 1492 Byte [PR88]. Beim Wireless LAN IEEE 802.11x beträgt der maximale Payload[3] sogar 2312 Byte [ANS07].

Jede Nachricht QuickLinks sollte nun vollständig in ein Paket passen. Die

[1] Siehe dazu auch Abschnitt 2.2.3 ab Seite 18.

[2] Die Dateneinheiten werden auf den einzelnen Schichten unterschiedlich bezeichnet: Auf der Netzwerkschicht heißen sie *frame* oder *Rahmen*, auf der Internetschicht *datagram* oder *Datagramm, Paket* (allgemein für alle Dateneinheiten verbindungslose Protokolle, so auch bei UDP), und auf der Transport- und Anwendungsschicht (bei verbindungsorientierten Protokollen) *message* oder *Nachricht*. An dieser Stelle wird aus Übersichtlichkeitsgründen der Begriff Paket für die Dateneinheiten aller Schichten verwendet.

[3] Dieser wird dort als Framebody bezeichnet.

kleinste maximale Paketgröße im Vergleich der betrachteten Protokolle des Protokollstapels wird also vom Ethernet bestimmt und bietet der nächst höheren Netzwerkschicht einen Payload von maximal 1492 Byte. Auf der IP-Schicht kommen nun noch 24 Byte für den IP-Header [Pos81a] $l_{IP-header}$ hinzu, so dass IP noch 1468 Byte dem nächst höheren Protokoll UDP zur Verfügung stellen kann. Der UDP-Header verlangt noch einmal 8 Byte [Pos80] ($l_{UDP-header}$) vom Paketvolumen, so dass auf der Schicht der Java-Anwendung QuickLink noch 1460 Byte für die maximale Nachrichtengröße N_{max} zur Verfügung stehen. Kann also die Größe aller QuickLink-Nachrichtenarten $N_{maintenance}$ und $N_{dynamic}$ mit 1460 Byte abgebildet werden

$$N_{maintenance} \leq N_{max} \text{ und } N_{dynamic} \leq N_{max},$$

so ist sichergestellt, dass diese jeweils über alle Netzwerkschichten hinweg als ein Paket im lokalen Netzwerk versendet werden.

Theoretisch mögliche, maximale Paketgröße einer Zyklusnachricht Die Größe der Zyklusnachricht in QuickLink kann über dessen Konfigurationsdatei variabel eingestellt werden. Sie ist dann aber bei jeder ausgeführten Instanz gleich groß und konstant groß. Es muss also beim Festlegen der Parameter in der Konfigurationsdatei darauf geachtet werden, dass Bedingung $N_{maintenance} \leq N_{max}$ eingehalten wird. Dies ist erfüllt, wenn die in Abschnitt 11.2.1.1 auf Seite 193 vorgestellte Formel zur Berechnung der Nachrichtengröße einer Zyklusnachricht die Bedingung

$$l_{header} + (max * (l_{address} + l_{services})) \leq N_{max}$$

einhält.

Unter den in Abschnitt 11.2.1.1 genannten (realistischen) Bedingungen mit einer IP-Adressenlänge $l_{address}$ von 4 Byte, 32 abbildbaren Diensten mit einer Länge $l_{services}$ von 4 Byte und einer QuickLink-List Headerlänge l_{header} von 5 Byte können dann maximal

$$max \leq (N_{max} - l_{header})/(l_{address} + l_{services})$$

QuickLink-Knoten abgebildet werden. Dies sind konkret

$$max \leq (1460 - 5)\ Byte/(4 + 4)\ Byte = 181,9\ ,$$

also 181 QuickLink-Knoten. Dieser Wert stellt unter den angenommenen Bedingungen die obere Schranke der verwaltbaren Knoten eines lokalen QuickLink-Netzwerkes dar.

Die sich daraus ergebende Ethernetpaketgröße $P_{maintenance}$ auf der Netzwerkschicht berechnet sich nun wieder aus der maximalen Nachrichtengröße einer Zyklusnachricht $N_{maintenance}$ zuzüglich der Header und Trailer der darunter liegenden Netzwerkschichten aus

$$P_{maintenance} = N_{maintenance} + l_{UDP-header} + l_{IP-header} + l_{Eth-header} + l_{Eth-trailer}.$$

Die Paketgröße $P_{maintenance}$ beträgt daher mit oben angenommenen Parametern und mit

$$N_{maintenance} = l_{header} + (max * (l_{address} + l_{services}))$$

konkret

$$P_{maintenance} = 5 + (181 * (4 + 4)) + 8 + 24 + 22 + 4\ Byte = 1511\ Byte.$$

Realistisch mögliche, maximale Paketgröße einer Zyklusnachricht Im vorherigen Abschnitt wurden die möglichen Paketgrößen anhand der Beschränkungen der Protokolle des Protokollstapels berechnet. In der aktuellen Implementierung QuickLinkNets verwendet QuickLink aber einen Byte-Wert zur Verwaltung der Anzahl der Knoten in der QuickLink-List. Da hierfür nur die positiven Zahlen des möglichen Wertebereichs des Datentyps `Byte` verwendet werden, ist die Anzahl der maximal verwaltbaren QuickLink-Knoten in einem lokalen QuickLink-Netzwerk auf 127 Knoten beschränkt. Insofern sind die im vorherigen Abschnitt berechneten Werte für die aktuelle prototypische Implementierung nicht maßgeblich, wohl aber für spätere Ausbaustufen QuickLinkNets.

Die maximale Nachrichtengröße $N_{maintenance}$ reduziert sich daher auf den in Abschnitt 11.2.1.1 auf Seite 11.2.1.1 bereits berechneten Wert von

$$N_{maintenance} = 1021\ Byte.$$

Unter diesen Voraussetzungen sinkt auch die realistische Paketgröße $P_{maintenance}$ auf

$$P_{maintenance} = 1079\ Byte.$$

Paketgröße einer Updatenachricht Wie im Abschnitt 11.2.1.2 dargestellt wurde, beträgt die konkrete Nachrichtengröße $N_{dynamic}$ für eine Updatenachricht unter den angenommenen Einstellungen konstant 13 Byte. Damit ist die Bedingung $N_{dynamic} \leq N_{max}$ in jedem Falle erfüllt. Die Größe des verbrauchten Payloads $l_{dynamic}$ eines Ethernetpakets berechnet sich aus

$$l_{dynamic} = N_{dynamic} + l_{UDP-header} + l_{IP-header}$$

und ergibt mit $N_{dynamic}$ = 13 Byte, $l_{UDP-header}$=8 Byte und $l_{IP-header}$=24 Byte lediglich

$$l_{dynamic} = 13 + 8 + 24\ Byte = 45\ Byte.$$

Rechnet man nun für das Ethernetpaket noch den Header mit einer Länge $l_{Eth-header}$ von 22 Byte und den Trailer $l_{Eth-trailer}$ von 4 Byte hinzu, ergibt sich eine Ethernetpaketgröße $P_{dynamic}$ für eine Updatenachricht von

$$P_{dynamic} = 45 + 22 + 4\ Byte = 71\ Byte.$$

11.2.2 Paketaufkommen

Nachdem nun die möglichen Größen der Pakete bestimmt sind, soll das Aufkommen, also wie oft ein Paket gesendet wird, betrachtet werden. Aus dem Aufkommen einer bestimmten Paketsorte in einer bestimmten Zeit kann deren Sendefrequenz, also die Häufigkeit einer Paketsendung in einer bestimmten Zeit, bestimmt werden.

11.2.2.1 Paketaufkommen der Zyklusnachrichten

Zyklusnachrichten werden von QuickLink in regelmäßigen Abständen gesendet. Nach dem Ablauf eines Zykluses sendet immer nur ein QuickLink-Knoten eine Zyklusnachricht. Aus der Anzahl der Zyklusnachrichten No_{cycle} pro Zykluszeit t_{cycle} ergibt sich die Sendefrequenz $F_{maintenance}$, die für die Wartung und den Zusammenhalt des QuickLink-Netzwerkes maßgeblich ist, aus

$$F_{maintenance} = No_{cycle}/t_{cycle}.$$

Die Sendezeit, d.h. der regelmäßige Zyklustakt, wird in der Konfigurationsdatei QuickLinks voreingestellt, wobei unsere Empfehlung für t_{cycle} bei einer Sekunde liegt. Eine Zykluszeit von einer Sekunde empfiehlt sich nach unserer Einschätzung, um die in einem hochdynamischen logischen Netzwerk notwendige Reaktionsgeschwindigkeit zu erreichen. Die zahlreichen messtechnischen Untersuchungen (siehe Abschnitte 11.3.2 und 11.3.3) am Prototypen QuickLink zeigen dies. Die konkrete Sendefrequenz $F_{maintenance}$ für Zyklusnachrichten betragen unter den genannten Bedingungen

$$F_{maintenance} = 1/1\,s = 1/s = 1\,Hz.$$

11.2.2.2 Paketaufkommen der Updatenachrichten

Im Gegensatz zu den deterministisch anfallenden Zyklusnachrichten für die reguläre Netzwerkverwaltung kann das Aufkommen $F_{dynamic}$, das durch die Beitritts- und Aktualisierungsnachrichten ausgelöst wird, nicht vorhergesagt werden. Dies liegt an der unvorhersagbaren Häufigkeit der Beitritts- und Aktualisierungsnachrichten, die von der vorherrschenden Netzwerkdynamik bestimmt wird und schwanken kann. Das Aufkommen $F_{dynamic}$ berechnet sich aus der im Zeitraum zwischen t_0 und t_1 anfallenden Nachrichtenanzahl m aus

$$F_{dynamic}(t_0, t_1) = m/(t_1 - t_0).$$

Die Häufigkeit der Beitritts- und Aktualisierungsnachrichten lässt sich nur schwer abschätzen. Unterstellt man ein hochdynamisches Netzwerk mit durchschnittlich zwei Eintritten und zwei Aktualisierungsvorgängen, also einem m von vier, in einem Zeitraum (t_0, t_1) von einer Sekunde, was wir für realistisch halten, so ent-

steht ein konkretes Aufkommen $F_{dynamic}$ von

$$F_{dynamic}(1s) = 4/1\ s = 4/\ s = 4\ Hz.$$

11.2.3 Resultierende Netzwerklast

Nachdem nun die maximalen Paketgrößen und die Aufkommen der Paketarten diskutiert wurden, kann die resultierende Netzwerklast L bestimmt werden. Sie berechnet sich allgemein aus

$$L = P * F.$$

11.2.3.1 Netzwerklast durch Zyklusnachrichten

Die konkret entstehende maximale Netzwerkbelastung $L_{maintenance}$, hervorgerufen durch die Zyklusnachrichten, ergibt sich demnach aus der maximalen Paketgröße $P_{maintenance}$ und Sendefrequenz $F_{maintenance}$ aus

$$L_{maintenance} = P_{maintenance} * F_{maintenance}.$$

Im durch den Protokollstapel vorgegebenen Fall mit den in den vorherigen Abschnitten berechneten Werten von $P_{maintenance} = 1511\ Byte$ und $F_{maintenance} = 1/s$, ergäbe sich eine maximale Netzwerklast für die Wartung des QuickLink-Netzwerkes $L_{maintenance}$ von

$$L_{maintenance} = 1511\ Byte * 1/s = 1511\ Byte/s \approx 1,48\ KByte/s.$$

Für die maximale Netzwerklast der aktuellen prototypischen Implementierung mit ihrer Knotenbegrenzung auf 127 beträgt die maximale Netzwerklast

$$L_{maintenance} = 1079\ Byte * 1/s = 1079\ Byte/s \approx 1,02\ KByte/s.$$

11.2.3.2 Netzwerklast durch Updatenachrichten

Die konkret entstehende Netzwerkbelastung $L_{dynamic}$, hervorgerufen durch die Beitritts- und Aktualisierungsnachrichten, ergibt sich aus der maximalen Paketgröße $P_{dynamic}$ und Sendefrequenz $F_{dynamic}$ aus

$$L_{dynamic} = P_{dynamic} * F_{dynamic}.$$

Unter den in den Abschnitten 11.2.1.3 und 11.2.2.2 getroffenen Annahmen einer Paketgröße $P_{dynamic}$ von 71 Byte und einer durchschnittlichen Sendefrequenz $F_{dynamic}$ von 4 Hz berechnet sich die durchschnittliche Netzwerklast $L_{dynamic}$ für dynamische Vorgänge im QuickLink-Netzwerk aus

$$L_{dynamic} = 71\ Byte * 4/s = 284\ Byte/s \approx 0,28\ KByte/s.$$

11.2.4 Diskussion Netzwerklast

Das logische QuickLinkNet-Netzwerk betrachtet die "Welt" aus zwei Blickwinkeln, welche das logische Netzwerk in eine lokale und eine globale Sicht unterteilen. Die Sichten beziehen sich zum einen auf das Infrastrukturelement Region, welches durch QuickLink und ServiceJuggler realisiert ist, und zum anderen auf das interregionale, globale Netzwerk, welches durch APLICOOVER in Verbindung mit P-Grid realisiert ist. Da für beide Bereiche unterschiedliche Eigenschaften gefordert wurden, sind sie auch mit unterschiedlichen Konzepten und Paradigmen implementiert worden.

Andere verteilte Systeme, wie z.B. File-Sharing-Systeme, beruhen i.d.R. auf einer einheitlichen Struktur und trennen den lokalen Bereich nicht vom globalen. Da sie folglich auch ein einheitliches Kommunikationsmuster verwenden, müssen sie, wenn sie sich über das globale Internet erstrecken wollen, Protokolle mit sicheren Verbindungseigenschaften verwenden. Dies erfolgt dann über das TCP-Protokoll. Eine Verwendung von echten Broadcastnachrichten[4] schließt sich damit aus. Obwohl auch andere Peer-to-Peer-Netzwerke trotz dieser Umstände lokale Bereiche bilden, können sie ihrerseits bei der dynamischen Netzwerklast nicht mit QuickLink mithalten: Je nach Struktur des Netzwerkes sind dazu bei n Netzwerkknoten meist zwischen $log(n)$ und $n * n - 1$ TCP-Verbindungen nötig, über welche dann doch nur zum Großteil redundante Daten, wie sie bei Aktualisierungen des Netzwerkes ausgetauscht werden müssen, übertragen werden. QuickLinks Nachrichtenaufwand ist dagegen durch den Einsatz von Broadcastnachrichten von der Anzahl der am Netzwerk teilnehmenden Knoten n unabhängig, daher ist der Nachrichtenaufwand konstant.

11.3 Praktische Evaluation des logischen Netzwerkes von QuickLink

Die in diesem Abschnitt beschriebenen Messexperimente und die Diskussion ihrer Ergebnisse dienen dem Nachweis der praktischen Anwendbarkeit und erwarteten Leistungsfähigkeit QuickLinks. QuickLink verwendet UDP-basierte Broadcastnachrichten zur Verwaltung des Netzwerkes. Da das Protokoll UDP nur unzuverlässige Verbindungseigenschaften leistet, ist das Ankommen der gesendeten Daten nicht garantiert. Bei der Implementierung QuickLinks wurden daher

[4] Einige verteilte Anwendungen verwenden sogenannte Broadcasts über ihre logische Infrastruktur, welche aber nur auf dem logischen Level der Anwendungen als solche bezeichnet werden können. Ein solcher "logischer Broadcast", gemeint ist das Versenden einer inhaltsgleichen Nachricht an mehrere Empfänger, muss spätestens auf der Transportschicht (OSI-Schicht 4/Internetschicht 3) durch entsprechend viele Unicast-Verbindungen umgesetzt werden.

schon Maßnahmen zur Geringhaltung der zu sendenden Paketgrößen[5] getroffen, um die Wahrscheinlichkeit der Störung des Netzwerkes klein zu halten. Die folgenden Messungen sollen nun die Robustheit der broadcastbasierten Kommunikation QuickLinks und die dabei entstehende, rechentechnische Belastung der QuickLink-Knoten verdeutlichen.

Dazu wird zuerst der Messaufbau und die verwendeten Geräte beschrieben, welche für alle experimentellen Evaluierungen des QuickLink-Netzwerkes verwendet wurden. Anschließend werden thematisch getrennt die einzelnen Experimente und das Vorgehen bei der Aufstellung der Messreihen beschrieben und die Ergebnisse diskutiert.

11.3.1 Messumgebungen

Die Messexperimente wurden in einem Ethernet nach IEEE 802.3u und in einem Wireless LAN nach IEEE 802.11b durchgeführt. Es wurde jeweils ein für sich isoliertes Testnetzwerk aufgebaut, welches als Messumgebung diente. Durch den isolierten Aufbau wurden Störungen durch fremden, nicht definierten Datenverkehr ausgeschlossen. Als Hardware der Messumgebungen dienten die in der folgenden Tabelle 11.1 aufgeführten Rechner und die in Tabelle 11.2 aufgeführten Netzwerkgeräte. Je nach Anforderungen wurden die Rechner und Netzwerkgeräte entsprechend konfiguriert und zu dem gewünschten Messaufbau kombiniert.

Um das Verhalten der QuickLink-Kommunikation zu testen, wurden bei den netzwerkbezogenen Messexperimenten jeweils ein QuickLink-Netzwerk etabliert und sein Verhalten aufgezeichnet. In mehreren Messreihen wurde das physikalische Netzwerk dann mit jeweils einem definierten Datenstrom beaufschlagt, um eine schrittweise steigende Auslastung des Netzwerkes zu erreichen. Der Datenstrom wurde durch zwei zusätzliche Rechner erzeugt, welche nicht zum logischen QuickLink-Netzwerk gehörten, aber dasselbe technische Netzwerk benutzten. Der Datenstrom selbst wurde durch die Übertragung einer sehr großen, gepackten Datei von einen Rechner auf den anderen mit Hilfe des Dienstes `Secure Copy (scp)`, einem in fast jedem unixbasierenden Betriebssystem verfügbaren Datenübertragungsdienst, erzeugt. Auf dem sendenden Rechner wurde der Datenstrom mit Hilfe des Tools `Traffic Control (tc)` [Hub07] begrenzt, wobei der Begrenzungswert über die Messreihen variiert wurde und als Parameter der Messungen diente.

11.3.1.1 Messaufbau Ethernet

Das erste Testnetzwerk war, wie in Abbildung 11.1 illustriert, ein Ethernet nach IEEE 802.3u Standard mit einer Bandbreite von 100 MBit/s im Betriebsmo-

[5] Vergleiche dazu auch Abschnitt 11.2.1.3 über sinnvolle Paketgrößen, die in QuickLink eingestellt werden können.

Name	Prozessortyp	MHz	RAM	Speicher-typ	Betriebssysteme
Armada	Intel Pentium II	266	192 MB	SD	Linux 2.4.19 / Win98 SE
Toshiba	Intel Pentium IV	1600	256 MB	SD	Linux 2.6.18 / WinXP Home
Maxdata	Intel Pentium M	1700	1024 MB	DDR1	Linux 2.6.16 / WinXP Prof. SP2
Gericom	Intel Pentium IV	2000	448 MB	SD	Linux 2.6.16 / WinXP Prof. SP2
Thinkpad	Intel T2500	2000	2048 MB	DDR2	Linux 2.6.16 / WinXP Prof. SP2
Silentbob	AMD Athlon64 3200+	2000	1024 MB	DDR1	Linux 2.6.16 (64Bit) / WinXP Prof. SP2
IPC677	Intel Pentium 4	2800	1024 MB	DDR1	Linux 2.6.16 / WinXP Prof. SP2

Tabelle 11.1: Einige technische Daten der verwendeten Messrechner

Name	Gerätefunktion	Ethernet	Wireless LAN
Netgear WGT624	Router / Switch / WLAN-AP	LAN: 4 Ports 802.3u, Uplink	802.11b/g
SMC 7004ABR	Router / Switch	LAN: 4 Ports 802.3u, Uplink	-

Tabelle 11.2: Einige technische Daten der verwendeten Netzwerkgeräte

dus Vollduplex. Die drei Rechner *Gericom*, *Silentbob* und *Maxdata* dienten als QuickLink-Knoten im Netzwerk. Die Rechner *Toshiba* und *Thinkpad* erzeugten bei den Messexperimenten, bei denen das Netzwerk definiert belastet werden sollte, den definierten Datenstrom. Dieser wurde auf *Toshiba* mittels `tc` begrenzt und von dort an *Thinkpad* gesendet.

Die netzwerktechnische Hardware, bestehend aus den in Tabelle 11.2 aufgeführten Ethernet Switch-Router-Kombinationen, wurden miteinander kaskadiert, um die notwendige Anzahl von Ethernetanschlüssen zu erlangen. Dabei wurden die drei Rechner für das QuickLink-Netzwerk und der den Datenstrom sendende Rechner *Toshiba* an den Router *Netgear* angeschlossen, während der den Datenstrom empfangende Rechner *Thinkpad* als einziger an den am Uplink vom *Netgear* angeschlossenen zweiten Router *SMC* hing.

Auf jedem der angeschlossenen Messrechner lief QuickLink in der gleichen

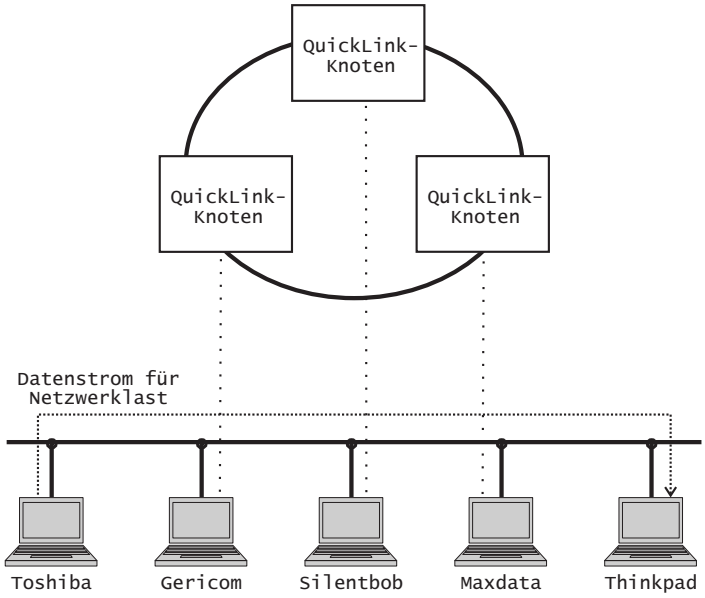

Abbildung 11.1: Der Messaufbau 1 für die Messungen im IEEE 802.3u Ethernet, 100 MBit/s, Vollduplex

Konfiguration. Die für die Messung verwendeten und ausschlaggebenden Parameter QuickLinks zeigt Tabelle 11.3.

QuickLink-Parameter	Bedeutung
usedIPAddressLength=4	Verwendung von IPv4 Adressen
sendTime=1000	Zykluszeit (cycle time) 1000 ms
offsetTime=200	Karenzzeit für den Empfang einer Zyklusnachricht 200 ms
headerlength=5	Headerlänge der QuickLink-List / Zyklusnachricht von 5 Byte
maxNumberOfServices=32	32 verwaltbare Dienste
maxNumberOfNodes=127	die maximale Anzahl zu verwaltbarer Knoten

Tabelle 11.3: Die bei den Netzwerkmessungen verwendeten Parameter QuickLinks

11.3.1.2 Messaufbau Wireless LAN

Das zweite Testnetzwerk zeigt Abbildung 11.2, ein Wireless LAN nach IEEE 802.11b Standard mit einer maximalen Bandbreite von 11 MBit/s im Infrastrukturmodus. Der Messaufbau ähnelt prinzipiell dem der Ethernetmessumgebung.

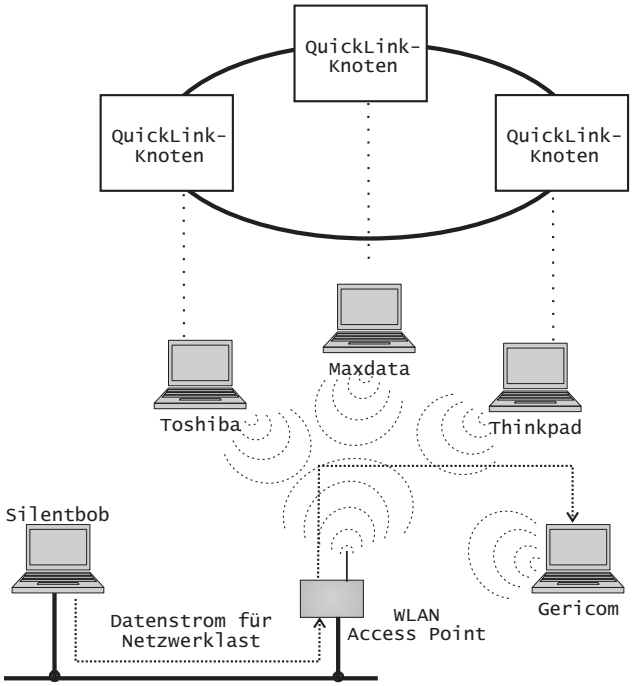

Abbildung 11.2: Der Messaufbau 2 für die Messungen im IEEE 802.11b Wireless LAN, 11 MBit/s, Infrastrukturmodus

Allerdings wurde als Netzwerk-Hardware nur der Router/WLAN-Access Point *Netgear* verwendet. Als Messrechner kamen *Toshiba*, *Maxdata* und *Thinkpad* zum Einsatz, welche das QuickLink-Netzwerk bildeten. Den definierten Datenstrom zur Netzwerkbelastung erzeugte der Messrechner *Silentbob*, der ihn über den Ethernetanschluss an den Access Point und dieser ihn über das Wireless LAN an *Gericom* schickte.

11.3.2 Zuverlässigkeit

11.3.2.1 Messmethodik

Das Ziel der Zuverlässigkeitsmessungen war der Nachweis, dass eine UDP-basierte Kommunikation in einem lokalen Netzwerk unter den Bedingungen

- geringe Paketgröße und, durch die relativ geringe Sendefrequenz, wie sie von QuickLink erzeugt wird,
- relativ geringe Netzwerkbelastung

ausreichend zuverlässig ist. Dazu wurde in beiden Testnetzwerken über die Messzeit ein QuickLink-Netzwerk etabliert. Während dieser Zeit zeichneten alle

QuickLink-Knoten in regelmäßigen Abständen die Anzahl ihrer wahrgenommenen QuickLink-Knoten auf. Im Falle eines ungestörten Ablaufes der Verwaltungsvorgänge in QuickLink, wenn also alle Zyklusnachrichten regelmäßig gesendet und von allen anderen QuickLink-Knoten empfangen werden, sollte die Anzahl der wahrgenommenen QuickLink-Knoten auf allen beteiligten Knoten gleich groß sein, konstant bleiben und der Anzahl der tatsächlich anwesenden QuickLink-Knoten entsprechen. Der Zeitpunkt, an dem die beteiligten Knoten die Anzahl der wahrgenommenen QuickLink-Knoten feststellen sollen, wurde im Verarbeitungsablauf eines QuickLink-Knotens auf den Zeitpunkt nach dem Empfang und der Verarbeitung einer Zyklusnachricht festgelegt, da mit dem Empfang der Zyklusnachrichten auch die zeitlichen Abläufe aller QuickLink-Knoten synchronisiert werden. Die Messungen wurden daher auf allen QuickLink-Knoten quasi simultan durchgeführt. Das folgende Stück Pseudocode illustriert die Messungen auf den QuickLink-Knoten:

Require: start time is taken
1: **while** current time < (start time + measurementTime) **do**
2: **if** cycle message is received **then**
3: process cycle message
4: print out current number of QuickLink-Nodes
5: **end if**
6: **end while**

11.3.2.2 Messreihen Ethernet

In dieser Konstellation wurden nun sieben Messreihen mit jeweils mindestens 1000 Messwerten erstellt. In jeder Messreihe wurde der Datenstrom zur Auslastung des Netzwerkes geändert, um dessen Einfluss auf die Zuverlässigkeit der Broadcastkommunikation QuickLinks herauszufinden.

Messreihe	Netzwerkbelastung begrenzt auf
1	ohne Begrenzung
2	80 MBit/s
3	50 MBit/s
4	20 MBit/s
5	10 MBit/s
6	5 MBit/s
7	ohne Begrenzung mit 3 Datenströmen

Tabelle 11.4: Die durchgeführten Messreihen im Ethernet nach IEEE 802.3u, 100 MBit, Vollduplex

In Tabelle 11.4 sind die durchgeführten Messreihen zusammengefasst dargestellt.

11.3.2.3 Messreihen Wireless LAN

Es wurden ebenfalls sieben Messreihen mit jeweils 1000 Messwerten erstellt. In jeder Messreihe wurde wieder der Datenstrom zur Auslastung des Netzwerkes geändert, um dessen Einfluss auf die Zuverlässigkeit der Broadcastkommunikation QuickLinks herauszufinden. In der nachfolgenden Tabelle 11.5 sind die durchgeführten Messreihen zusammengefasst dargestellt. Die Messreihe 7 entspricht

Messreihe	Netzwerkbelastung begrenzt auf
1	ohne Begrenzung
2	10 MBit/s
3	8 MBit/s
4	5 MBit/s
5	3 MBit/s
6	1 MBit/s
7	10 MBit/s mit abweichender QL-Konfiguration

Tabelle 11.5: Die durchgeführten Messreihen im Wireless LAN nach IEEE 802.11b, 11 MBit, Infrastrukturmodus

der Messreihe 2, allerdings mit abgeänderter QuickLink-Konfiguration. Um den Einfluss der Zykluszeit auf die Robustheit der QuickLink-Kommunikation zu testen, wurden, abweichend von den in Tabelle 11.3 geltenden Bedingungen, die Zykluszeit t_{cycle} auf 2000 ms (sendTime=2000) und die Karenzzeit $t_{waiting}$ auf 300 ms (offsetTime=300) angehoben.

11.3.2.4 Ergebnisse und Diskussion

Messreihen Ethernet Die Messreihen zu den Zuverlässigkeitsmessungen im Ethernet lieferten sehr gute Ergebnisse. Das zentrale Diagramm über alle Messreihen und Messrechner ist in Abbildung 11.3 zu sehen. Dort sind die Zustände der Messrechner, ausgedrückt durch den arithmetischen Mittelwert der wahrgenommenen Anzahl der QuickLink-Knoten, über der Bandbreitenbeschränkung des das Netzwerk belastenden Datenstromes aufgezeigt. Die Mittelwerte sind mit Errorbars für die Minimalwerte versehen, welche ausdrücken, ob der betreffende Rechner während der jeweiligen Messreihe vorhandene QuickLink-Knoten nicht erkannte. Die Ursache für die Nichterkennung konnte in der isolierten Messumgebung nur in einer nicht empfangenen Zyklusnachricht liegen, welche wiederum

durch Kollision mit einem Datenpaket des das Netzwerk belastenden Datenstromes verursacht wurde. Die Errorbar drücken allerdings nur das Auftreten eines solchen Zustandes aus, nicht aber die Häufigkeit desselben, welche aber aus der absoluten Höhe des Mittelwertes abgelesen werden kann.

Im Idealfall, wenn also keine "Ausfälle" von QuickLink-Knoten detektiert werden würden, gäbe es keine Errorbars und der Mittelwert würde genau drei betragen, denn so viele QuickLink-Knoten waren während jeden Messexperimentes permanent vorhanden. Die zum Diagramm in Abbildung 11.3 zugehörigen Werte Messwerte können in Tabelle 11.6 nachgelesen werden.

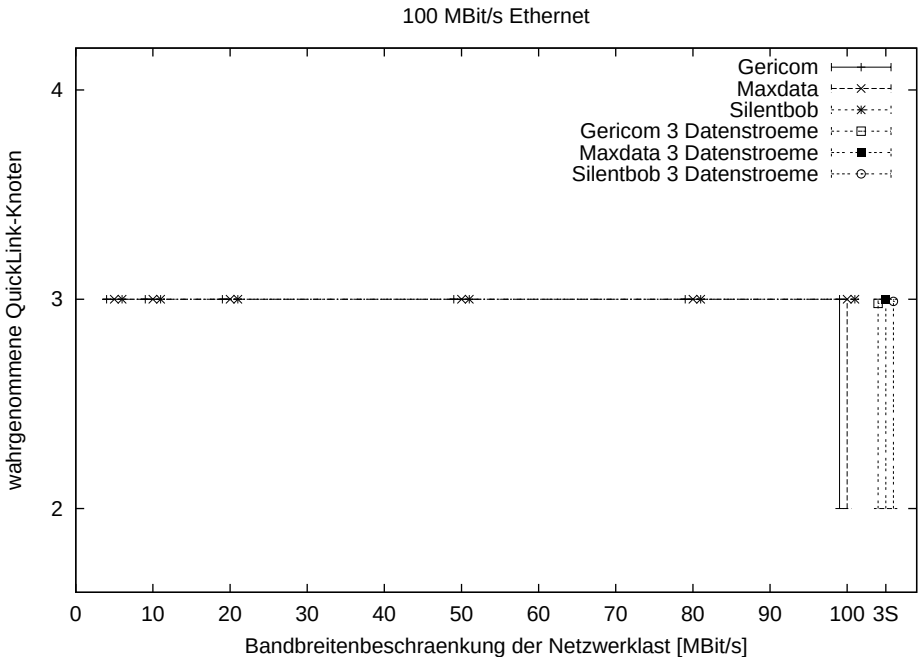

Abbildung 11.3: Die arithmetischen Mittelwerte der Zuverlässigkeitsmessungen im Ethernet in Abhängigkeit von der Netzwerkbelastung; der Wert $3S$ auf der X-Achse steht für die Messreihe 7 mit drei unbegrenzten Datenströmen bei 100 MBit/s und ist daher nicht mit der Linie, welche die übrigen Messreihen verbindet, verbunden worden

Wie an den Messwerten zu erkennen ist, zeigt sich die QuickLink-Kommunikation im Ethernet als sehr robust. Während der Messreihen, bei dem das Netzwerk durch einen begrenzten Datenstrom belastet wird, werden zu jedem Zeitpunkt alle vorhandenen QuickLink-Knoten erkannt. Dies lässt darauf schließen, dass überhaupt keine Ausfälle von QuickLink-Nachrichten auftreten.

Erst mit dem unbegrenzten Datenstrom bei 100 MBit/s treten Ausfälle bei den Messrechnern *Gericom* und *Maxdata* auf. Hier sind also einzelne Zyklus-

Netzwerklast in MBit/s	Gericom		Maxdata		Silentbob	
	Avg	Min	Avg	Min	Avg	Min
5	3	3	3	3	3	3
10	3	3	3	3	3	3
20	3	3	3	3	3	3
50	3	3	3	3	3	3
80	3	3	3	3	3	3
100	3	2	3	2	3	3
100 3 Ströme	2,98	2	3	2	2,99	2

Tabelle 11.6: Die zu den im Diagramm in Abbildung 11.3 abgebildeten Messreihen gehörenden Werte, alle Angaben in Anzahl QuickLink-Knoten

nachrichten zwar gesendet worden, aber scheinbar nicht angekommen, so dass der sendende Knoten als nicht mehr vorhanden detektiert wurde. Ein Blick auf die Messreihen der drei Messrechner im Vergleich in Abbildung 11.4 zeigt, dass die Ausfälle während der ca. 1000 Messungen nur zweimal auftraten. Der sendende Rechner *Silentbob* hat seine eigene Nachricht sehr wohl empfangen, da diese im Speicherbereich des eigenen Netzwerkprotokollstapels vom Bereich "Ausgang" in den Bereich "Eingang" kopiert wurde und nicht über das Netzwerk empfangen werden musste. *Silentbob* erkennt daher zu diesen Zeitpunkten drei QuickLink-Knoten, während die anderen Messrechner, bei denen die über das Netzwerk geleitete Nachricht nicht ankommt, nur zwei QuickLink-Knoten erkennen.

Aufgrund der guten Messwerte auch bei einer Netzwerkbelastung mit unbegrenztem Datenstrom wurden in der letzten Messreihe 7 nun 3 Datenströme gleichzeitig etabliert, und zwar wie bisher von *Toshiba* an *Thinkpad* und zusätzlich je ein Datenstrom von *Toshiba* an *Gericom* und an *Silentbob*. Die beiden Messrechner wurden deshalb gewählt, weil sie leistungsmäßig den schwächsten (*Gericom*) und den stärksten QuickLink-Knoten (*Silentbob*) in der Messumgebung darstellten. Diese extreme Belastung verursachte bei den QuickLink-Knoten *Gericom* und *Silentbob* eine neben der QuickLink-Kommunikation wirkende, zusätzliche Belastung des Prozessors, des Datenbusses und der Festplatte sowie der Netzwerkhardware und -software. Die Ergebnisse dieser absichtlichen Überlastung der Messumgebung ist im Diagramm in Abbildung 11.3 ebenfalls zu sehen: Wie die Errorbars zeigen, verzeichnen alle drei Messrechner Ausfälle.

Ein Blick auf die drei Messreihen im Vergleich in Abbildung 11.5 auf Seite 209 gibt einen genaueren Aufschluss über die Verhältnisse im Netzwerk unter diesen Bedingungen: *Gericom* hat mit Abstand die meisten Ausfälle. Dass *Silentbob* dagegen wesentlich besser aussieht, liegt sicherlich an seiner deutlich besseren Gesamtperformance. Maxdata als einziger unbelasteter Messrechner hat während

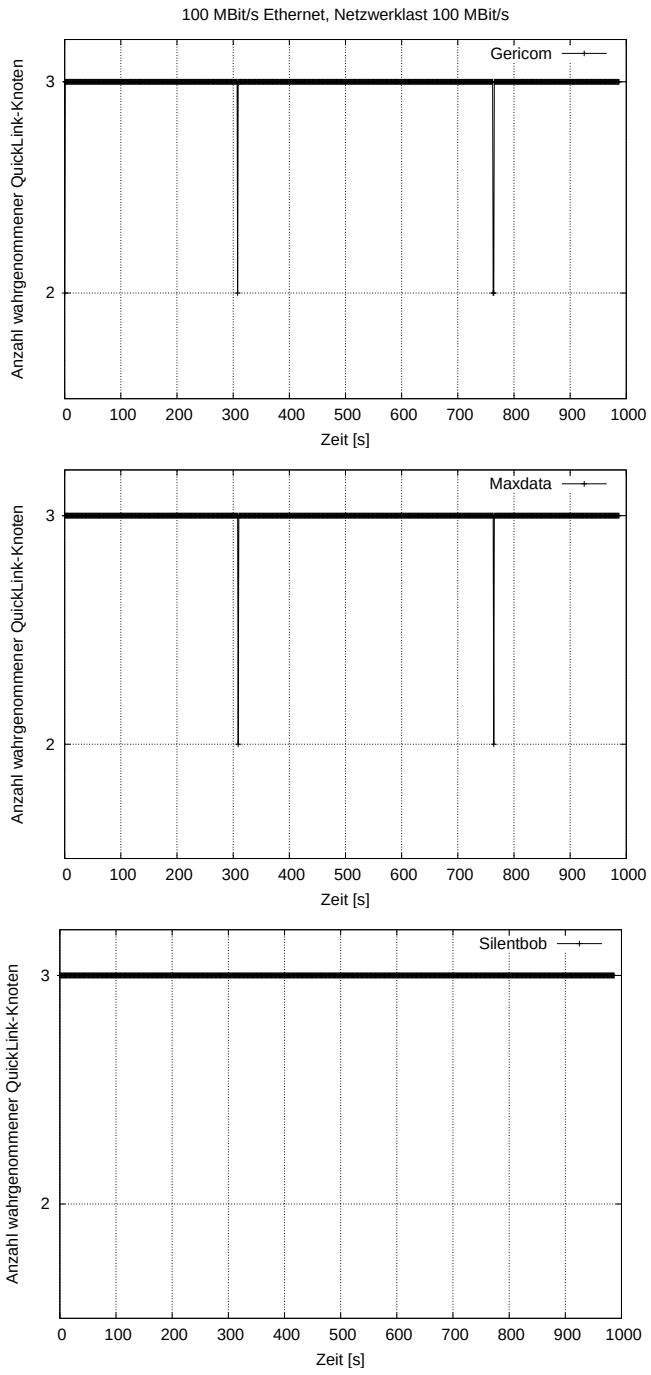

Abbildung 11.4: Die Messreihen der Zuverlässigkeitsmessungen aller drei Messrechner in einem Ethernet mit einem belastenden Datenstrom ohne Bandbreitenbegrenzung

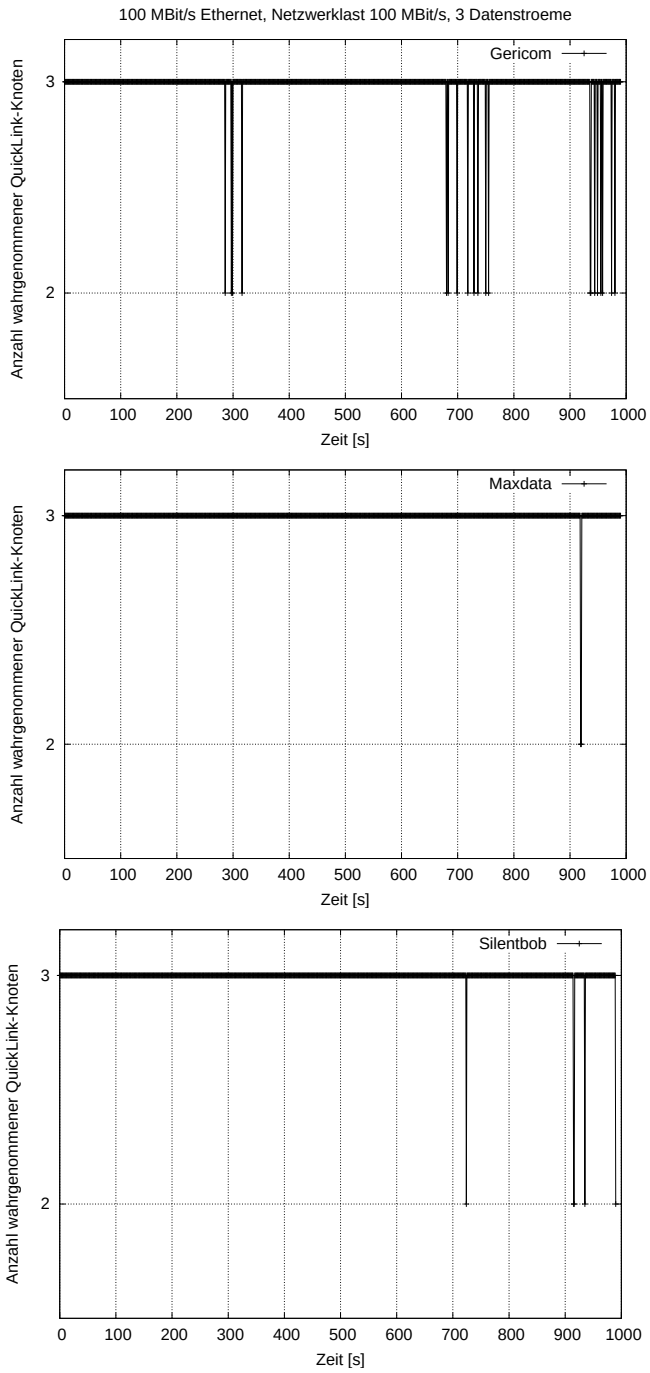

Abbildung 11.5: Die Messreihen der Zuverlässigkeitsmessungen aller drei Messrechner in einem Ethernet mit drei belastenden Datenströmen ohne Bandbreitenbegrenzung

der gesamten Messreihe nur einen einzigen Ausfall zu verzeichnen, der sicherlich auf ein ausgefallenes Paket zurückzuführen ist. Da Silentbob und Maxdata zur gleichen Zeit, bei der bei Gericom Ausfälle auftreten, nur einzelne bzw. gar keine Ausfälle haben, müssen zu diesen Zeiten die QuickLink-Nachrichten über das Netzwerk übertragen worden sein. Dies ist ein weiterer Hinweis darauf, dass sich der Messrechner Gericom unter diesen extremen Bedingungen an seiner Lastgrenze bezüglich der Rechenleistungszuteilung an die QuickLink-Software befand.

Die Zuverlässigkeit der QuickLink-Kommunikation in einem Ethernet kann als Ergebnis der Messungen als sehr gut angesehen werden.

Messreihen Wireless LAN Die Messungen im Wireless LAN ließen keine so guten Messwerte wie im Ethernet erwarten. Wireless LAN besitzen auch im unbelasteten Zustand gegenüber kabelgebundenen Ethernets eine deutlich höhere Latenzzeit, was sich im direkten Vergleich durch eine gewisse Trägheit beim Aufbau von Netzwerkverbindungen bemerkbar macht.

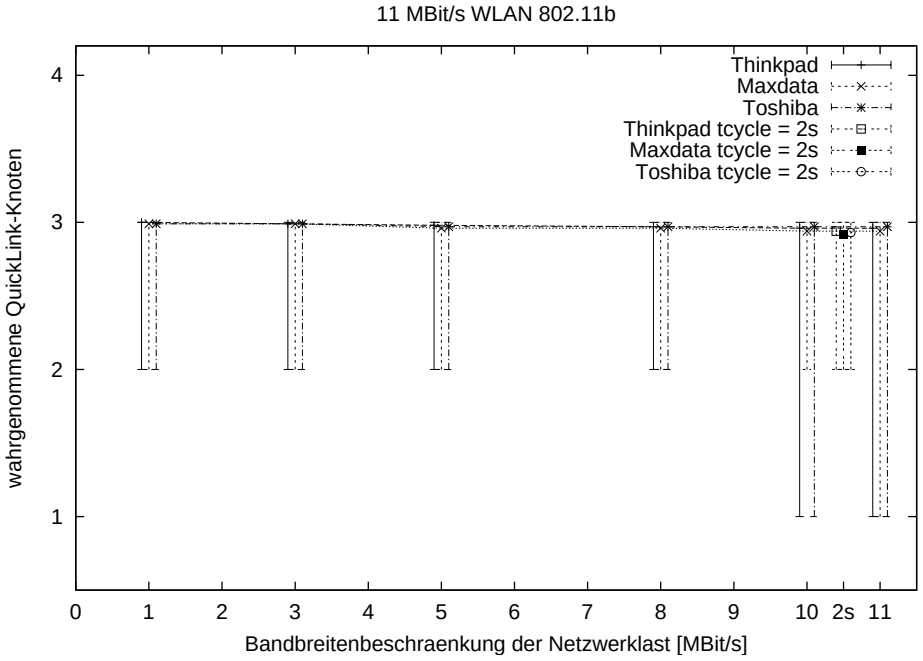

Abbildung 11.6: Die arithmetischen Mittelwerte der Zuverlässigkeitsmessungen im Wireless LAN in Abhängigkeit von der Netzwerkbelastung; der Wert $2s$ auf der X-Achse steht für die Messreihe 7 mit einem auf 10 MBit/s begrenzten Datenstrom, einer t_{cycle} von 2 Sekunden und einer Karenzzeit $t_{waiting}$ von 300 Millisekunden

Die Abbildung 11.6 zeigt das zentrale Diagramm aller Messreihen, in dem die arithmetischen Mittelwerte der während der Messreihen wahrgenommenen QuickLink-Knoten aller Messrechner zusammengefasst dargestellt sind. Die (nur negativen) Errorbars kennzeichnen wieder das Auftreten von nichtdetektierten QuickLink-Knoten ihrer absoluten Höhe nach. Zur besseren Übersichtlichkeit sind die Messwerte der drei Messrechner auf der X-Achse nebeneinander dargestellt, die Achsenwerte entsprechen also einer Kategorie, in welche die Messwerte aller drei Messrechner fallen. Die genauen Werte lassen sich aus der zugehörigen Tabelle 11.7 entnehmen.

Netzwerklast in MBit/s	Thinkpad		Maxdata		Toshiba	
	Avg	Min	Avg	Min	Avg	Min
1	3	2	2.99	2	2.99	2
3	2.99	2	2.99	2	2.99	2
5	2.98	2	2.96	2	2.97	2
8	2.97	2	2.96	2	2.97	2
10	2.96	1	2.94	2	2.97	1
11	2.96	1	2.94	1	2.97	1
10, $t_{cycle} = 2s$, $t_{waiting} = 300ms$	2,94	2	2,92	2	2,93	2

Tabelle 11.7: Die zu den im Diagramm in Abbildung 11.6 abgebildeten Messreihen gehörenden Werte, alle Angaben in Anzahl QuickLink-Knoten

Im Gegensatz zum Ethernet treten die "Ausfälle" von Knoten schon bei der geringen Netzwerkbelastung von 1 MBit/s auf und nehmen mit zunehmender Belastung mengenmäßig etwas zu. Dies drückt sich im sinkenden Mittelwert der erkannten QuickLink-Knoten aus, der allerdings nur anhand der Werte in Tabelle 11.7 genauer zu erkennen ist. Ab 10 MBit/s Netzwerkbelastung treten sogar Situationen auf, in denen bis zu zwei Knoten nicht erkannt werden.

Die Charakteristik des Verhaltens im Wireless LAN zeigt ein Vergleich der Messreihen. Stellvertretend für alle drei Messrechner dient hier *Thinkpad*: In Abbildung 11.7 sind seine Zeitreihen für 1, 5, 8, 10 und 11 MBit/s Belastung sowie bei 10 MBit/s bei 2 Sekunden Zykluszeit und 300 ms Karenzzeit gegenübergestellt. Gut zu sehen ist die absolute Zunahme der "Ausfälle" und auch die Höhe der gleichzeitig nicht erkannten QuickLink-Knoten mit zunehmender Netzwerkbelastung.

Aufgrund der Messwerte wurde in Messreihe 7 bei einer Netzwerklast von 10 MBit/s die Zykluszeit t_{cycle} auf 2 Sekunden und die Karenzzeit $t_{waiting}$ auf 300 ms heraufgesetzt, um den Einfluss der QuickLink-Parameter auf die Qualität der QuickLink-Verbindung unter den Bedingungen im Wireless LAN herauszu-

finden. Aus dem zu dieser Messreihe zugehörigen Diagramm unten rechts in Abbildung 11.7 lässt sich erkennen, dass sich die Höhe der gleichzeitig nicht erkannten QuickLink-Knoten von 2 auf 1 reduziert. Trotzdem ist es erstaunlich, dass sich dafür die Menge der nicht erkannten Knoten insgesamt erhöht. Ein Blick in die Tabelle 11.7 bestätigt dies, da der arithmetische Mittelwert dieser Messreihen bei allen Messrechnern den geringsten Wert aufweist. Die Erhöhung der Zykluszeit hat unter den Bedingungen des Messaufbaus scheinbar nur eine relativ geringe Auswirkung auf die Verbesserung der Verbindungseigenschaften QuickLinks.

Die Ursache kann im vom Wireless LAN verwendeten Zugriffsverfahren *Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA)* [Wik07b] liegen, welches ein Wettbewerbsverfahren realisiert. Ein sendewilliger Knoten hört zuerst das Medium ab und prüft, ob es frei ist. Ist es belegt, wartet er eine bestimmte, zufällige Zeit (Interframe Space (IFS)) ab und prüft dann erneut die Verfügbarkeit. Dieser Ablauf wird so lange mit sich ändernden Wartezeiten fortgeführt, bis das Medium im Augenblick der Verfügbarkeitsprüfung frei ist. Erst jetzt kann der sendewillige Knoten anfangen zu senden. Mit dem Senden belegt der Sender das Netzwerk für eine gewisse Zeit *exklusiv*. Alle anderen sendewilligen Knoten müssen dann wieder eine gewisse Zeit warten, um das Medium auf Verfügbarkeit zu überprüfen. Leider kann bei diesem einfachen CSMA/CA-Verfahren weder die Kollisionsfreiheit noch eine zeitliche Schranke für den Medienzugriff garantiert werden. Wenn ein Sender nun einen ständigen Datenstrom über den Access Point an einen Empfänger sendet und deshalb der Sender das Netzwerk für relativ lange Zeitabschnitte exklusiv belegt, kann dies in der Praxis zu erheblichen Verzögerungen beim Losschicken anstehender Sendungen anderer Knoten kommen. Dies führt zu den teilweise erheblichen Latenzzeiten im Wireless LAN. Um diese Vermutung zu unterlegen, wurde noch eine Messung in der Wireless LAN Messumgebung bei voller Netzwerklast, aber ohne QuickLink-Netzwerk, durchgeführt. Über den PING-Dienst des Betriebssystems wurde die Roundtripzeit *(Round-Trip-Time (RTT))* von einem unbelasteten Knoten zu dem den Belastungsdatenstrom sendenden Knoten, dem diesen empfangenden Knoten und zu einem anderen, am Datenstrom unbeteiligten Knoten gemessen.

Die Ergebnisse dieser Messreihe sind im oberen Diagramm und die Mittelwerte mit Errorbars im unteren Diagramm in Abbildung 11.8 zu sehen. Die Werte für die *Round-Trip-Time*, d.h. für ein Hin und Her der Ping-Nachricht, liegen durchschnittlich zwischen ca. 70 ms beim empfangenden und ca. 145 ms beim sendenden Knoten. Der unbelastete Knoten liegt in der Mitte, bei ca. 120 ms. Diese Werte erscheinen sehr hoch, vor allem im Vergleich zur Karenzzeit QuickLinks, die bei 200 ms liegt.

Die Abweichungen vom Mittelwert nach oben sind in ihrer Anzahl und Intensität ebenfalls erheblich. Sie erreichen und überschreiten die Karenzzeit von

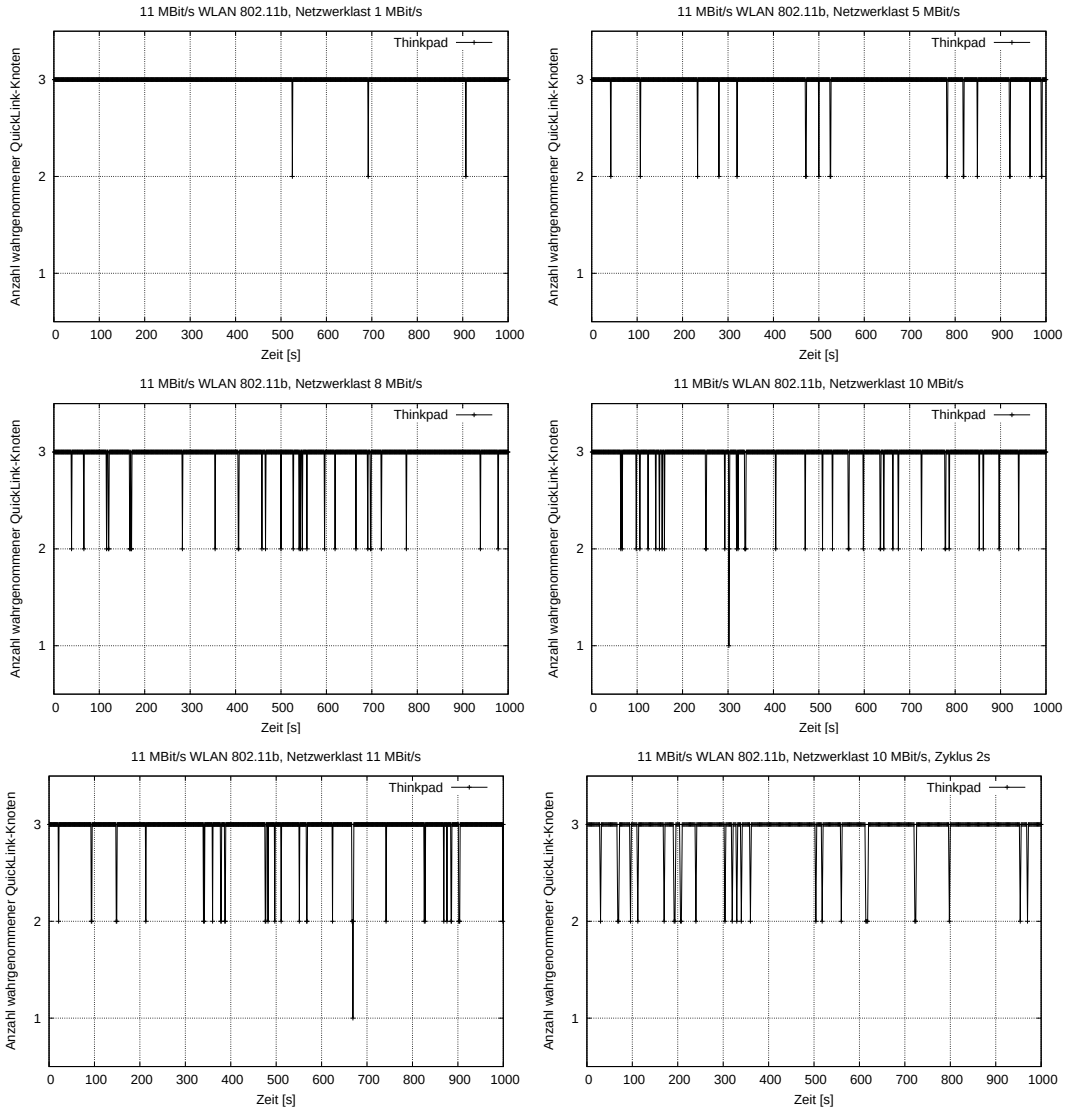

Abbildung 11.7: Die Zeitreihen des Messrechners *Thinkpad* bei einer Belastung von 1, 5, 8, 10 und 11 MBit/s sowie bei 10 MBit/s und bei $t_{cycle} = 2\,s$ und $t_{waiting} = 300\,ms$

200 ms regelmäßig, was zwangsläufig zu einer Nichtberücksichtigung der in solchen Situationen gesendeten Zyklusnachrichten führen muss. Insofern ist die Verbesserung bei der Messreihe 7, bei der die Ausfälle auf einen Knoten gleichzeitig verringert werden konnten, dem Einfluss der Erhöhung der Karenzzeit auf 300 ms zuzuschreiben und nicht dem Einfluss der Erhöhung der Zykluszeit.

Ein weiterer interessanter Aspekt der Messwerte ist, dass das größte Maximum

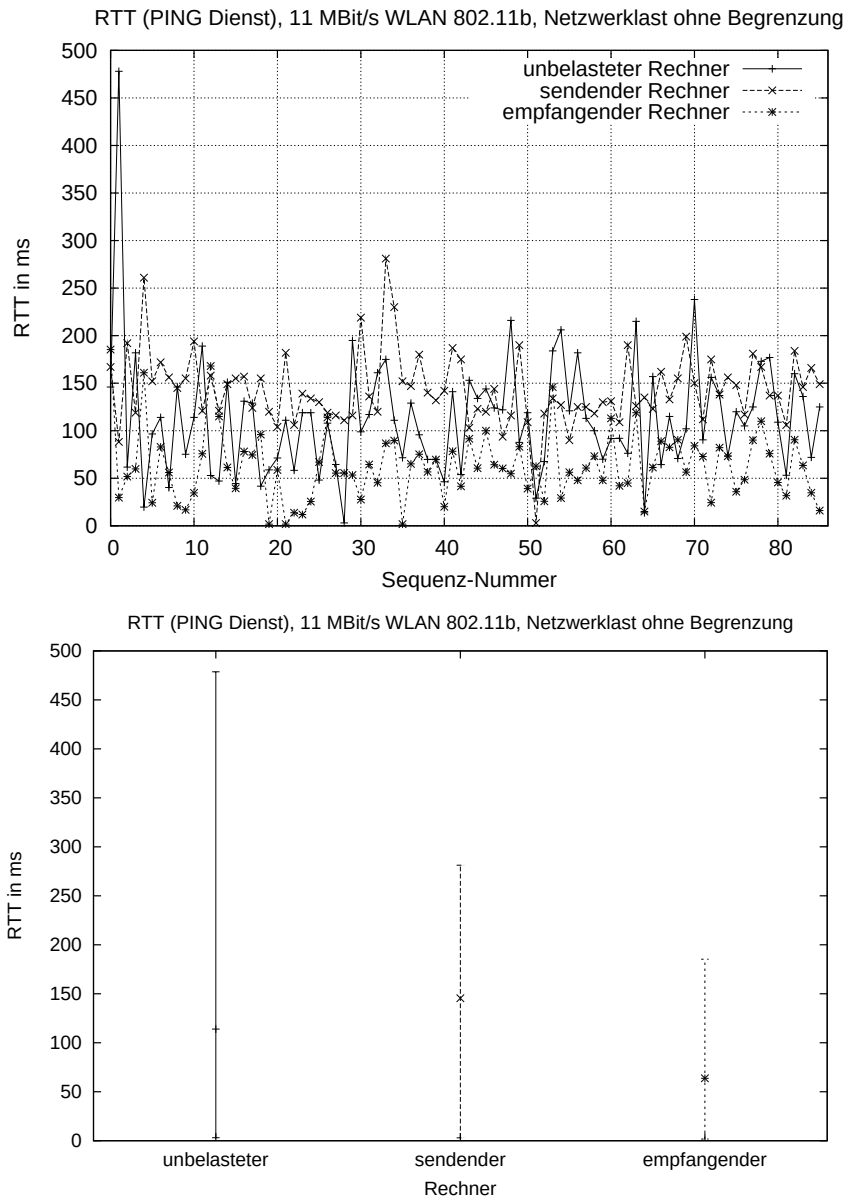

Abbildung 11.8: Die Zeitreihen der PING-Messreihe im Wireless LAN (oben) und deren kumulierte Auswertung (unten)

der RTT beim unbelasteten Knoten zu finden ist, was die Aussage stützt, dass die Wartezeiten beim CSMA/CA für zeitweise am Senden unbeteiligte Knoten steigt, wenn noch ein anderer, regelmäßiger Netzwerkverkehr vorhanden ist. Da sich ein Messrechner in Messreihe 7 mit der auf zwei Sekunden erhöhten Zy-

kluszeit quasi wie ein unbeteiligter Knoten verhält, schließlich will er bei drei QuickLink-Knoten nur alle sechs Sekunden senden, kommt hier der beschriebene Effekt des CSMA/CA-Zugriffsverfahrens voll zum Tragen und wirkt kontraproduktiv: Ein verspätetes Senden führt im QuickLink-System zum Ignorieren der Nachricht durch die empfangenden QuickLink-Knoten, auf welchen die Karenzzeit schon abgelaufen ist, und damit zum Rauswurf des sendenden Knotens aus dem QuickLink-Netzwerk. Die insgesamt höchste Zahl an nichterkannten QuickLink-Knoten, wie sie an den niedrigen Mittelwerten aller Messrechner (siehe Tabelle 11.7) für die Messreihe 7 ersichtlich sind, bestätigen die Auswirkungen dieses Effektes auf das QuickLink-Netzwerk. Als Gegenmaßnahme könnte unter diesen Bedingungen die Karenzzeit zukünftig für alle QuickLink-Knoten auf bis zu 400 ms erhöht und die Zykluszeit bei einer Sekunde beibehalten werden, um eine weitere Verbesserung zu erhalten.

Da herausgeworfene QuickLink-Knoten dies innerhalb einer Zykluszeit detektieren, melden sie sich durch das Senden einer Updatenachricht, die zwischen den Zyklen gesendet wird, sofort wieder an. Insofern sind die Ausfälle im QuickLink-Netzwerk, wie sie auch in den Zeitreihen in Abbildung 11.7 abgelesen werden können, nur als sporadisch anzusehen, denn die herausgefallenen Knoten sind meist innerhalb einer Zykluszeit wieder eingebunden. Die "3 QuickLink-Knoten"-Linie drückt den Normalzustand des QuickLink-Netzwerkes in den Messreihen-aus, wenn also alles in Ordnung ist. Alle Zeitreihen zeigen zwar Ausfälle, sind aber auf ca. 1000 Messwerte bezogen, welche durch die enge Drängung der einzelnen Messpunkte auf der "3 QuickLink-Knoten"-Linie nur als dicke Linie zu erkennen ist. Aus dem Verhältnis der Ausfälle zum normalen Zustand des QuickLink-Netzwerkes kann man den Gesamteffekt abschätzen. Grundsätzlich ist der Zusammenhalt des QuickLink-Netzwerkes auch unter diesen Bedingungen gegeben.

11.3.3 Der Eintritt ins Netzwerk

11.3.3.1 Messmethodik

Die Messexperimente für die Eintrittsmessungen im Ethernet und im Wireless LAN basieren auf den im Abschnitt 11.3.1 beschriebenen Messaufbauten und den QuickLink-Einstellungen, wie sie in Tabelle 11.3 angegeben wurden. Sie entsprechen also den Messbedingungen wie bei den Zuverlässigkeitsmessungen. Ziel der Messungen war es, die Eintrittsgeschwindigkeit eines QuickLink-Knotens unter verschiedenen Netzwerkauslastungen und in verschiedenen Netzwerktypen zu ermitteln. Dazu wurde ein QuickLink-Netzwerk etabliert, in welches ein neuer QuickLink-Knoten versucht einzutreten. Der neue QuickLink-Knoten war dazu so präpariert, dass er beim Versuch, in das Netzwerk einzutreten, die Zyklustakte anhand der eingetroffenen Zyklusnachrichten zählte, bis er vollständig eingebun-

den war. Dies war der Fall, wenn eine Zyklusnachricht empfangen wurde, in der seine eigenen Daten enthalten waren und sich die Anzahl der im Netzwerk vorhandenen QuickLink-Knoten auf die für das Testnetzwerk in dieser Konstellation maximale Anzahl von 3 erhöht hatte. Im besten Falle kann der Eintritt also eine Zykluszeit dauern. Nach seinem erfolgreichen Eintritt ins QuickLink-Netzwerk entfernte sich der eingetretene QuickLink-Knoten selbst wieder aus dem Netzwerkverbund, blieb dann eine Wartezeit von 5 t_{cycle} untätig, um die Detektion seines Austritts durch die verbliebenen QuickLink-Knoten sicherzustellen, und versuchte dann den Eintritt erneut. Der folgende Pseudocode beschreibt das Verhalten des eintrittswilligen Knotens:

Require: maxNumberOfNodes = 3
Require: state = ISOLATED
Require: waitingTime = 5*cycleTime
Require: i = 0

```
1:  while i < 1000 do
2:    state := JOINING
3:    send update message
4:    while currentNumberOfNodes < maxNumberOfNodes do
5:      j := 1
6:      if cycle message is received then
7:        if currentNumberOfNodes < maxNumberOfNodes then
8:          j++
9:        else
10:         print out "i, j"
11:       end if
12:     end if
13:   end while
14:   i++
15:   state := ISOLATED
16:   ignore all network traffic
17:   sleep for waitingTime
18: end while
```

Als eintrittswilliger QuickLink-Knoten wurde für die beiden verwendeten Messumgebungen 1 und 2 der Messrechner *Maxdata* präpariert. In dieser Konstellation wurden in beiden Messumgebungen jeweils sieben Messreihen mit jeweils 1000 Messwerten erstellt. In jeder Messreihe wurde der Datenstrom zur Netzwerkbelastung geändert, um deren Einfluss auf die Eintrittsgeschwindigkeit QuickLinks herauszufinden. Die durchgeführten Messreihen im Ethernet entsprechen denen in Tabelle 11.4 und die Messreihen im Wireless LAN denen in Tabelle 11.5.

11.3.3.2 Ergebnisse und Diskussion

Messreihen Ethernet Die Ergebnisse der Messreihen zu den Eintrittsmessungen im Ethernet sind im zentralen Diagramm über alle Messreihen und Messrechner in Abbildung 11.9 zu sehen. Dort sind die arithmetischen Mittelwerte der für den Eintritt in QuickLink-Netzwerk notwendigen QuickLink-Zyklen des Messrechners *Maxdata* über dem durch die auf der X-Achse aufgeführten Begrenzungen des Lastdatenstromes im technischen Netzwerk aufgetragen. Die jeweiligen Errorbars in Y-Richtung zeigen die zum Mittelwert der Messreihe gehörenden minimal und maximal benötigten QuickLink-Zyklen an, die zum Eintritt benötigt wurden. Die zugehörigen Zahlenwerte kann man Tabelle 11.8 entnehmen.

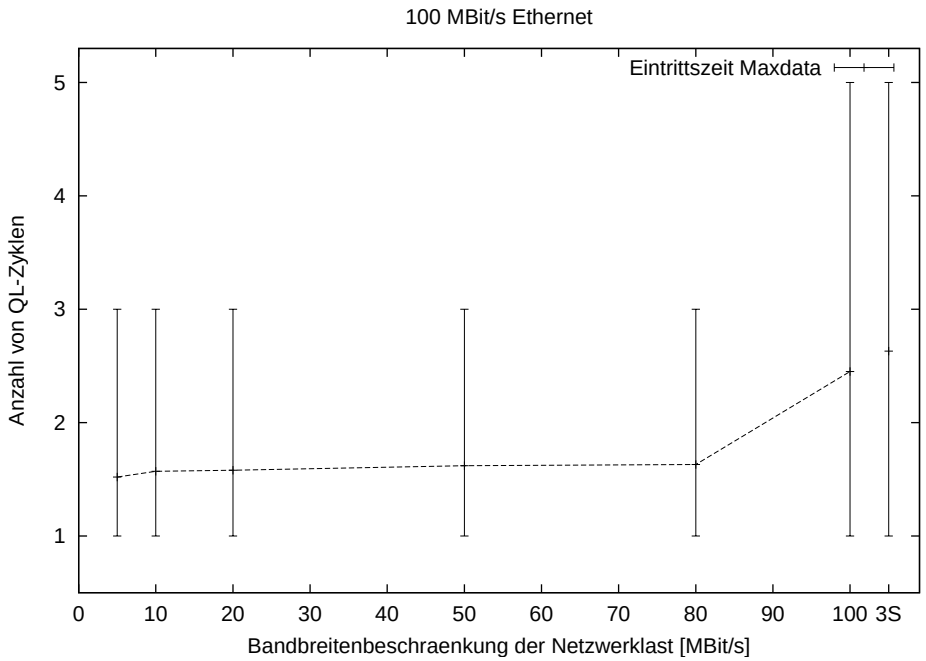

Abbildung 11.9: Die arithmetischen Mittelwerte der Eintrittsmessungen im Ethernet in Abhängigkeit von der Netzwerkbelastung; der Wert $3S$ auf der X-Achse steht für die Messreihe 7 mit *drei* unbegrenzten Datenströmen bei 100 MBit/s und ist daher nicht in die alle Messreihen verbindende Linie aufgenommen worden

Die Messwerte zeigen, dass der Messrechner *Maxdata* bis zu einer Netzwerkbelastung von 80 MBit/s maximal drei QuickLink-Zyklen benötigt, um dem Netzwerk beizutreten. Dabei steigt der arithmetische Mittelwert, der die durchschnittlich zu erwartende Zeitspanne bis zum Eintritt widerspiegelt, mit steigender Netzwerkbelastung leicht an, was mit dem zunehmenden störenden Einfluss

Netzwerklast in MBit/s	Maxdata		
	Avg	Min	Max
5	1,52	1	3
10	1,57	1	3
20	1,58	1	3
50	1,62	1	3
80	1,63	1	3
100	2,45	1	5
100 3 Ströme	2,63	1	5

Tabelle 11.8: Die zu den im Diagramm in Abbildung 11.9 abgebildeten Messreihen gehörenden Werte, alle Angaben in Anzahl QuickLink-Zyklen

des belastenden Datenstroms auf das Netzwerk korrespondiert. Er liegt aber insgesamt mit 1,52 bei 5 MBit/s bis 1,63 Zyklen bei 80 MBit/s Belastung auf geringem Niveau, wenn man bedenkt, dass das erreichbare Minimum zum Eintritt ins QuickLink-Netzwerk ein Zyklus beträgt.

Erst bei einer stärkeren Belastung als 80 MBit/s steigt die Eintrittzeit an, sie beträgt bei 100 MBit/s durchschnittlich 2,45 Zyklen und mit derselben Belastung mit drei Datenströmen im Netzwerk bei 2,63 Zyklen. Dieser etwas stärkere Anstieg der benötigten Zyklen in den beiden letzten Messreihen ist auf die zunehmend schlechteren Übertragungsbedingungen im Ethernet zurückzuführen. Dies kommt im Maximalwert von nunmehr fünf Zyklen zum Ausdruck, welche *Maxdata* in den beiden letzten Messreihen maximal benötigt, um dem QuickLink-Netzwerk beizutreten, und welche natürlich auch den arithmetischen Mittelwert anheben. Eine ähnliche Charakteristik zeigen auch die Ergebnisse der Zuverlässigkeitsmessungen im Ethernet, bei denen eine deutlich messbare Verschlechterung der Verbindungseigenschaften QuickLinks erst bei über 80 MBit/s Netzwerkbelastung einsetzt.

Die Eintrittsmessungen erlauben die Aussage, dass unter den Bedingungen der Messumgebung und einer QuickLink-Konfiguration mit einer Zykluszeit von einer Sekunde die typische Eintrittzeit eines QuickLink-Knotens im Ethernet zwischen einer und zwei Sekunden liegt und im schlechtesten Fall bei voll belastetem Netzwerk nach fünf Sekunden erledigt sein sollte.

Messreihen Wireless LAN Die Eintrittsmessungen im Wireless LAN ließen, nach den Erfahrungen mit den Zuverlässigkeitsmessungen im Ethernet und im Wireless LAN, ebenfalls etwas schlechtere Messwerte im Vergleich zum Ethernet erwarten. Diese Vermutung bestätigt sich mit Blick auf das zentrale Diagramm aller Messreihen in Abbildung 11.10, in dem die arithmetischen Mittelwerte der

benötigten QuickLink-Zyklen und ihre Minimum- und Maximumwerte als Errorbars gezeigt werden. Die genauen Werte lassen sich aus der zugehörigen Tabelle 11.9 entnehmen.

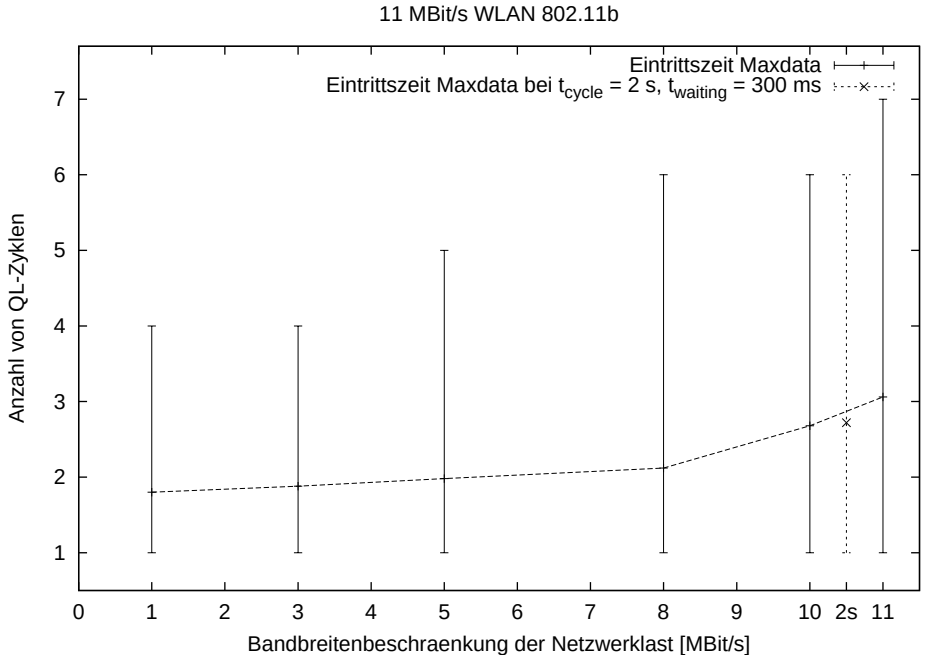

Abbildung 11.10: Die arithmetischen Mittelwerte der Eintrittsmessungen im Wireless LAN in Abhängigkeit von der Netzwerkbelastung; der Wert $2s$ auf der X-Achse steht für die Messreihe 7 mit einem auf 10 MBit/s begrenzten Datenstrom, einer t_{cycle} von 2 Sekunden und einer Karenzzeit $t_{waiting}$ von 300 Millisekunden

Obwohl die arithmetischen Mittelwerte aller Messreihen etwas höher liegen als bei den in der relativen Netzwerkbelastung vergleichbaren Messreihen im Ethernet, liegen sie mit zwischen durchschnittlich 1,8 Zyklen bei 1 MBit/s und durchschnittlich 3,06 Zyklen bei 11 MBit/s Belastung insgesamt in einem ähnlich guten Bereich. Auch der stärkere Anstieg des arithmetischen Mittelwertes ab etwa 80% Netzwerkauslastung, die beim Ethernet bei 80 MBit/s und beim Wireless LAN bei etwa 8 MBit/s erreicht wird, ähnelt der Anstiegscharakteristik im Ethernet (vergleiche Abbildung 11.9 auf Seite 217).

Im Vergleich zu den Ethernetmessungen fallen aber die in allen Messreihen und allen vergleichbaren Belastungszuständen des Netzwerkes hohen Maximalwerte auf, welche bei einer Bandbreitenbeschränkung bis 3 MBit/s bei 4 QuickLink-Zyklen liegen, bei 5 MBit/s auf 5 QuickLink-Zyklen, ab 5 MBit/s auf 6 Zyklen

Netzwerklast in MBit/s	Maxdata		
	Avg	Min	Max
1	1,80	1	4
3	1,88	1	4
5	1,98	1	5
8	2,12	1	6
10	2,68	1	6
11	3,06	1	7
10, $t_{cycle} = 2s$, $t_{waiting} = 300ms$	2,72	1	6

Tabelle 11.9: Die zu den im Diagramm in Abbildung 11.10 abgebildeten Messreihen gehörenden Werte, alle Angaben in Anzahl QuickLink-Zyklen

steigen und bei 11 MBit/s sogar 7 Zyklen erreichen. Einen Hinweis darauf, dass diese hohen Werte nur sporadisch auftreten, liefern die im Vergleich dazu niedrigen arithmetischen Mittelwerte in Abbildung 11.10 und in Tabelle 11.9.

Der Effekt der höheren Eintrittszeiten im Wireless LAN lässt sich zum einen durch das Verhalten des Zugriffsverfahrens CSMA/CA erklären, wie dies schon bei den Zuverlässigkeitsmessungen im Wireless LAN in Abschnitt 11.3.2.4 ab Seite 212 diskutiert wurde. Allerdings sollten sich die Auswirkungen des verzögerten Netzwerkzugriffs im Wireless LAN auf die Updatenachrichten nicht so stark auswirken wie auf die Zyklusnachrichten, da erstere keine zeitlichen Schranken im QuickLink-System einhalten müssen und quasi "irregulär", also zu fast beliebigen Zeitpunkten gesendet werden können. Zum anderen muss man aber auch bedenken, dass die hier beschriebenen Eintrittsmessungen in QuickLink-Zyklen gemessen wurden, welches durch das Zählen der eingehenden Zyklusnachrichten bis zum erfolgreichen Eintritt des eintrittswilligen Knotens ins Netzwerk geschah. Insofern überlagern sich bei dieser Art von Messungen im QuickLink-Netzwerk die Effekte, die durch Zyklusnachrichten entstehen und die bei den Zuverlässigkeitsmessungen beschrieben wurden, mit den Effekten, die direkt beim Eintritt eines QuickLink-Knotens durch die Updatenachrichten entstehen. Einen Beleg dafür liefert auch die Messreihe 7 mit veränderten QuickLink-Parametern. Die Veränderung der QuickLink-Parameter Zykluszeit und Karenzzeit wirken sich laut Implementierung nicht direkt auf die Updatenachricht aus, so dass der Einfluss dieser Parameter auf den Eintritt eines QuickLink-Knotens ins Netzwerk keine Rolle spielen sollte, wie dies Messreihe 7, im Diagramm in Abbildung 11.10 auf der X-Achse mit $2s$ markiert, auch deutlich zeigt. Der Eintritt mit der höheren Zykluszeit von zwei Sekunden und einer Karenzzeit von 300 ms erfolgt sogar etwas langsamer als bei der Vergleichsmessung von Messreihe 5 bei 10 MBit/s

mit normalen QuickLink-Parametern; allerdings sind die Abweichungen der Ergebnisse der Messreihen 5 und 7 voneinander, wie in Tabelle 11.9 zu sehen, gering (2,68 zu 2,72 Zyklen). Die entstandenen Abweichungen sind eher dem im vorherigen Absatz erwähnten Einfluss der QuickLink-Parameter auf die Zyklusnachrichten und damit auf den Zusammenhalt des QuickLink-Netzwerkes, wie in Abschnitt 11.3.2.4 diskutiert, zurückzuführen. Insofern wirkt sich der Einfluss der QuickLink-Parameter Zykluszeit und Karenzzeit indirekt schon geringfügig auf die Eintrittszeit aus, wie die geringen Unterschiede der Messreihen 5 und 7 belegen.

Abschließend kann man davon ausgehen, dass unter den Bedingungen der Messumgebung und mit einer Zykluszeit von einer Sekunde die typische Eintrittszeit eines QuickLink-Knotens über ein Wireless LAN zwischen zwei und drei Sekunden liegt und dass der Eintritt im schlechtesten Fall, bei voll belastetem Netzwerk, nach sieben Sekunden erreicht sein sollte.

11.3.4 Rechenzeit

Ziel dieser Messungen war es, den zeitlichen Rechenaufwand von QuickLink-Knoten für die Verarbeitung von QuickLink-Nachrichten zu ermitteln, um deren Belastung durch den ständigen Nachrichtenverkehr einschätzen zu können.

11.3.4.1 Messaufbau

Die Bestimmung der Rechenzeit für die Verarbeitung der QuickLink-Nachrichten ist vom Typ des Netzwerkes und der Belastung desselben mit Datenströmen unabhängig. Es kommt hierbei nur auf die Verarbeitungsgeschwindigkeit einer Nachricht innerhalb der QuickLink-Software auf dem QuickLink-Knoten an. Daher wurde, abweichend vom Messaufbau 1 für die Zuverlässigkeits- und Eintrittsmessungen, ein anderer Messaufbau 3 verwendet, wie er in Abbildung 11.11 skizziert ist.

Die Messungen fanden in einem Ethernet nach IEEE 802.3u im Vollduplexbetrieb statt, in welchem ein QuickLink-Netzwerk mit den vier Messrechnern *Gericom*, *Toshiba*, *Thinkpad* und *Silentbob* aufgebaut wurde.

11.3.4.2 Messreihen und Messmethodik

Es wurden vier Messexperimente mit jeweils 1000 Messwerten durchgeführt, bei denen die Anzahl der vorhandenen QuickLink-Knoten von 1 bis 4 variiert wurde, um den Einfluss der Anzahl der QuickLink-Knoten auf die benötigte Verarbeitungzeit herauszufinden. Bei jeder Messreihe wurde, wie in Tabelle 11.10 dargestellt, ein QuickLink-Knoten hinzugefügt.

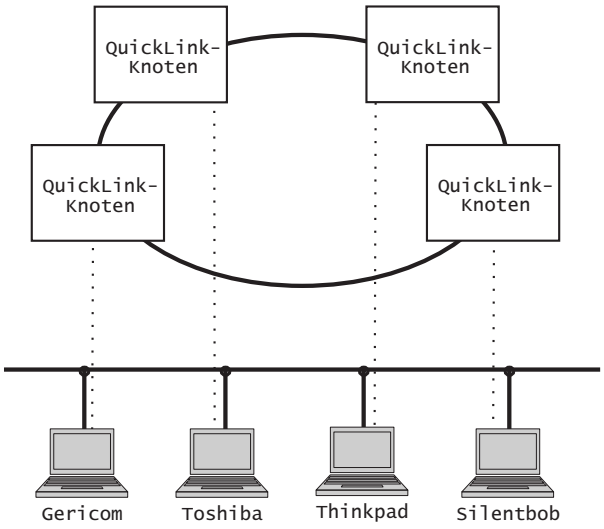

Abbildung 11.11: Der Messaufbau 3 für die Bestimmung der Verarbeitungszeiten von QuickLink-Nachrichten

Messreihe	Anzahl QuickLink-Knoten	beteiligte Messrechner
1	1	Silentbob
2	2	Silentbob, Thinkpad
3	3	Silentbob, Thinkpad, Gericom
4	4	Silentbob, Thinkpad, Gericom, Toshiba

Tabelle 11.10: Die durchgeführten Messreihen zur Rechenzeit

Während jeder Messreihe kommunizierten die QuickLink-Knoten mit Hilfe von Zyklus- und Updatenachrichten miteinander. Es wurde gelegentlich die Netzwerkverbindung eines Rechners getrennt, um den Ausfall und den Wiedereintritt eines Rechners zu simulieren und das Aufkommen von Updatenachrichten zu forcieren. Zyklus- und Updatenachrichten wurden während der Aufnahme der Messreihen simultan gemessen.

Für jede empfangene Zyklus- und Updatenachricht wurden zwei Zeiten gemessen, die für jeweils einen Messabschnitt stehen: Zum einen die Zeit vom vollständigen Empfang der Nachricht bis vor Beginn der Verarbeitung und zum anderen die Zeit für die Verarbeitung der Nachricht und die Aktualisierung der QuickLink-Listen. Die Summe beider Werte ergibt jeweils die Gesamtverarbeitungszeit. Der Hintergrund für die getrennte Aufnahme der beiden Messabschnitte ist die Erwartung, dass sich zumindest der erste Messabschnitt pro Messrechner konstant verhalten sollte, da der Aufwand für die Weiterleitung einer immer

gleich großen Nachricht auch konstant ist. Beim zweiten Messabschnitt ist aber eine Abhängigkeit von der Anzahl der QuickLink-Knoten im Netzwerk zu erwarten, wenn bei der Verarbeitung der Nachrichteninhalte die QuickLink-Liste abgearbeitet werden muss.

11.3.4.3 Ergebnisse und Diskussion

Rechenaufwand in Abhängigkeit von der Knotenzahl - Zyklusnachrichten Die Messreihen lassen Aussagen hinsichtlich der Verarbeitungszeit von Zyklus- und Updatenachrichten in Abhängigkeit von der Anzahl der QuickLink-Knoten im Netzwerk zu. Stellvertretend für alle Messrechner werden hier die Werte des Messrechners *Silentbob* betrachtet, da er in allen vier Messreihen (siehe Tabelle 11.10) anwesend war und somit als einziger Messrechner Werte für QuickLink-Netzwerke aller untersuchten Größen liefern konnte.

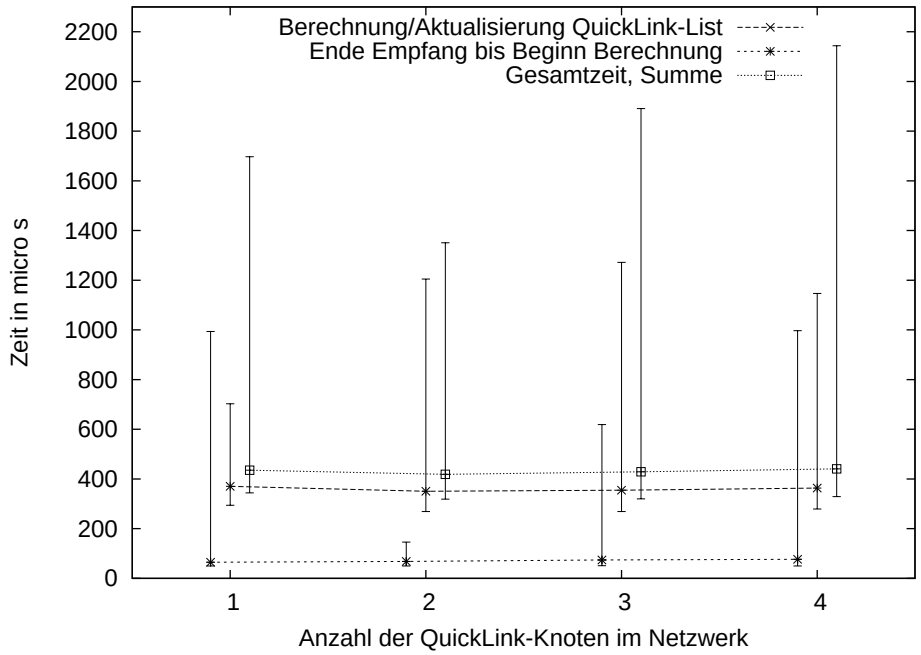

Abbildung 11.12: Die Verarbeitungszeiten von *Zyklusnachrichten* auf dem Messrechner *Silentbob* in Abhängigkeit von der Anzahl der QuickLink-Knoten im Netzwerk

Das Diagramm in Abbildung 11.12 zeigt die zeitlichen Aufwände *Silentbobs* zur Bearbeitung von Zyklusnachrichten in Abhängigkeit von der Knotenanzahl. Die zugehörigen Messwerte können ebenfalls aus der zugehörigen Tabelle 11.11 entnommen werden. Das Diagramm zeigt die arithmetischen Mittel der Messwer-

te mit ihren Minimum- und Maximumwerten als Errorbars. Die zwei Messabschnitte "Ende Empfang bis Beginn Berechnung" und "Berechnung/Aktualisierung QuickLink-List" sowie die Summe "Gesamtzeit, Summe" beider Messwerte sind auf der X-Achse um die X-Werte[6] herum etwas versetzt angeordnet worden, um eine bessere Ablesbarkeit zu ermöglichen. Alle Messwerte liegen auf einem auffallend niedrigen, gleichbleibenden Niveau. Die Zeit für den Aufwand

Anzahl QL-Knoten	zeitlicher Messabschnitt	Avg	Min	Max	Med
1	Empfang bis Beginn Berechnung	65,2	50	994	62
	Berechnung/Aktualisierung QuickLink-List	370,4	294	703	367
	Gesamtzeit (Summe)	435,6	344	1697	429
2	Empfang bis Beginn Berechnung	68,2	50	146	66
	Berechnung/Aktualisierung QuickLink-List	350,4	269	1205	347
	Gesamtzeit (Summe)	418,6	319	1351	413
3	Empfang bis Beginn Berechnung	74,2	51	619	68
	Berechnung/Aktualisierung QuickLink-List	354,8	269	1272	333
	Gesamtzeit (Summe)	429	320	1891	401
4	Empfang bis Beginn Berechnung	77	50	997	69
	Berechnung/Aktualisierung QuickLink-List	363,6	279	1147	360
	Gesamtzeit (Summe)	440,6	329	2144	429

Tabelle 11.11: Die Messwerte zur Verarbeitung von Zyklusnachrichten, wie sie im Diagramm in Abbildung 11.12 dargestellt sind; alle Angaben in Mikrosekunden

"Empfang bis Beginn Berechnung", die nach dem vollständigen Empfang einer Nachricht gestartet und mit dem Beginn der Verarbeitung der Nachricht gestoppt wurde, sollte bei einer immer gleichbleibenden Nachrichtengröße auf ein und demselben Rechner konstant bleiben. Die Messwerte bestätigen dies, obwohl die Abweichungen nach oben im Verhältnis zu den arithmetischen Mittelwerten

[6] Die X-Werte sind hier diskret, da es sich um die Anzahl der QuickLink-Knoten im QuickLink-Netzwerk handelt.

und den Medianwerten recht groß werden kann. Die Abweichungen lassen sich auf die Threadsteuerung Javas zurückführen, welche den internen Transport der Nachricht im QuickLink-System aufhalten kann. Dies geschieht, wenn z.B. der Thread zum Empfangen und Übergeben der Nachricht unterbrochen wird und die Rechenzeit einem anderen Thread zugeteilt wird oder, wenn der die Nachricht verarbeitende Thread im Moment der Übergabe noch keine Rechenzeit zugeteilt bekommt, um die Nachricht entgegenzunehmen. In diesem Falle wird die Messung noch nicht beendet, sondern die "Wartezeit" des übernehmenden Threads mitgemessen.

Die Zeit für den Aufwand "Berechnung/Aktualisierung QuickLink-List" verhält sich laut den Messwerten ebenfalls gleichbleibend. Die Verarbeitung der QuickLink-Liste ist so implementiert, dass beim Empfang einer neuen Zyklusnachricht die alte Liste immer komplett durch die neue Liste ersetzt wird. Dieser Vorgang ist deterministisch und erklärt die nahezu gleichbleibenden Messwerte. Wenn inhaltliche Unterschiede zwischen der alten und der neuen QuickLink-Liste detektiert werden, werden die konkreten Änderungen festgestellt und entsprechende Reaktionen QuickLinks, insbesondere Benachrichtigungen anderer Komponenten, angestoßen. Insofern sind die Maximalwerte als Abweichungen von den arithmetischen Mittelwerten gut als Situationen erklärbar, in denen eine inhaltliche Bearbeitung beider QuickLink-Listen notwendig war. Ferner hat an dieser Stelle auch wieder die Threadsteuerung Javas einen Einfluss auf die Messzeiten, da diese den Verarbeitungsthread jederzeit unterbrechen kann, zumal QuickLink und das ganze QuickLinkNet-Framework eine Multithread-Implementierung darstellt.

Insgesamt gesehen liegen die arithmetischen Mittel der Gesamtaufwände zur Verarbeitung von Zyklusnachrichten auf *Silentbob* zwischen 418,6 und 440,6 Mikrosekunden und haben damit einen sehr geringen absoluten Wert im Vergleich zu den die Zusammenarbeit der QuickLink-Knoten bestimmenden Zeitabläufen. So beträgt die Zykluszeit immerhin eine Sekunde und die Karenzzeit zum Empfang einer Zyklusnachricht 200 Millisekunden. Selbst der größte kumulierte Maximalwert liegt bei 2144 Mikrosekunden, was gegenüber 200 Millisekunden gerade einmal 1 Prozent ausmacht. Der Verarbeitungsaufwand für Zyklusnachrichten kann daher als sehr gering angesehen werden.

Rechenaufwand in Abhängigkeit von der Knotenzahl - Updatenachrichten Die Messwerte über den Aufwand zur Verarbeitung von Updatenachrichten auf *Silentbob* sind im Diagramm in Abbildung 11.13 und in Tabelle 11.12 dargestellt. Die Messwerte der einzelnen Messabschnitte sind zur besseren Ablesbarkeit auf der X-Achse wieder versetzt angebracht. Das gesamte Skalenniveau liegt etwas niedriger als in Abbildung 11.12 und weist auf die insgesamt geringeren Aufwände zur Verarbeitung von Updatenachrichten gegenüber Zyklusnachrichten hin.

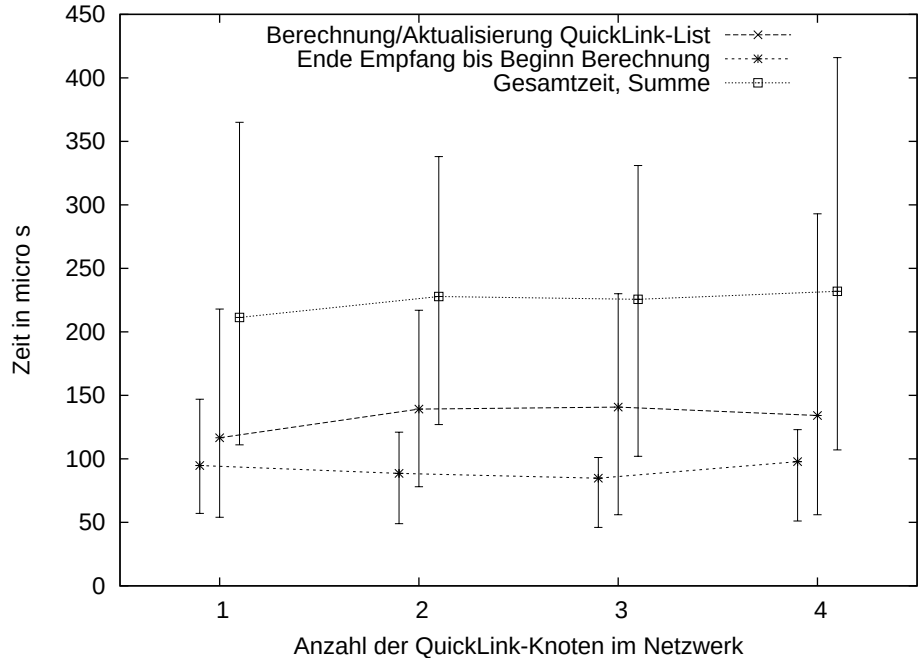

Abbildung 11.13: Die Verarbeitungszeiten von *Updatenachrichten* auf dem Messrechner *Silentbob* in Abhängigkeit von der Anzahl der QuickLink-Knoten im Netzwerk

Es fällt jedoch im direkten Vergleich der Messwerte auf, dass der Aufwand für die Updatenachricht im Messabschnitt "Empfang bis Beginn Berechnung" etwas höher liegt als bei der Zyklusnachricht und zwar um etwa 20 Mikrosekunden. Obwohl die Nachrichtengröße einer Updatenachricht erheblich kleiner als die der Zyklusnachricht ist, scheint diese Größe auf die Verarbeitungszeit in diesem Messabschnitt keinen Einfluss zu haben. Beide Receiver sind in QuickLink mit der gleichen Threadpriorität belegt und sollten daher auch nicht unterschiedlich behandelt werden. Der zeitliche Unterschied kann nur durch die Threadsteuerung Javas hervorgerufen werden, die einem nur sporadisch Rechenzeit verlangenden Thread, wie dies für den Updatereceiver in der Messumgebung, in welcher nur gelegentlich Updatenachrichten ankamen, zutraf, scheinbar auch entsprechend spät bedient.

Die Verarbeitungszeit für Updatenachrichten im Messabschnitt "Berechnung/ Aktualisierung QuickLink-List" verhält sich hingegen wie erwartet: Die arithmetischen Mittel der zeitlichen Aufwände sind hier deutlich geringer als bei der Verarbeitung einer Zyklusnachricht im gleichen zeitlichen Messabschnitt; sie liegen zwischen 116,6 und 140,8 Mikrosekunden und damit etwa 210 Mikrosekunden niedriger. Durch den großen Einfluss des Messabschnitts "Berechnung/Aktua-

Anzahl QL-Knoten	zeitlicher Messabschnitt	Avg	Min	Max	Med
1	Empfang bis Beginn Berechnung	94,7	57	147	96
	Berechnung/Aktualisierung QuickLink-List	116,6	54	218	78
	Gesamtzeit (Summe)	211,3	111	365	174
2	Empfang bis Beginn Berechnung	88,6	49	121	93
	Berechnung/Aktualisierung QuickLink-List	139,2	78	217	108
	Gesamtzeit (Summe)	227,8	127	338	201
3	Empfang bis Beginn Berechnung	84,8	46	101	96
	Berechnung/Aktualisierung QuickLink-List	140,8	56	230	143
	Gesamtzeit (Summe)	225,6	102	331	239
4	Empfang bis Beginn Berechnung	97,8	51	123	108,5
	Berechnung/Aktualisierung QuickLink-List	134,1	56	293	127
	Gesamtzeit (Summe)	231,9	107	416	235,5

Tabelle 11.12: Die Messwerte zur Verarbeitung von Updatenachrichten, wie sie im Diagramm in Abbildung 11.13 dargestellt sind; alle Angaben in Mikrosekunden

lisierung QuickLink-List" auf die Gesamtverarbeitungszeit liegen diese Werte ebenfalls auf niedrigem Niveau gegenüber den Werten für die Zyklusnachrichten, nämlich zwischen 211,3 und 231,9 Mikrosekunden. Das Niveau der Verarbeitungszeit einer Updatenachricht liegt also etwa bei der Hälfte der Verarbeitungszeit für eine Zyklusnachricht und ist damit sehr gering.

Eine Abhängigkeit der Verarbeitungszeit einer Updatenachricht von der Menge der QuickLink-Knoten lässt sich unter den Bedingungen der Messumgebung ebenfalls nicht feststellen, obwohl dies zu erwarten war. Schließlich wird beim Empfang von Updatenachrichten die aktuelle QuickLink-Liste der Reihe nach durchsucht und die neue Information durch Manipulation der QuickLink-Liste eingearbeitet. Allerdings hört die Suche spätestens beim Erreichen des ersten leeren Eintrags in der Liste auf. Daher lässt sich der Effekt, der eine im Mittel lineare Abhängigkeit der Bearbeitungszeit von der Menge der vorhande-

nen QuickLink-Knoten vermuten lässt, in einer mit nur vier Knoten gefüllten QuickLink-Liste nicht nachweisen. Ferner spricht dies für die sehr effiziente Implementierung der QuickLink-Listen als Byte-Arrays, welche ausschließlich durch einfache mathematische Operationen auf Grunddatentypen manipuliert werden.

Die maximalen Abweichungen aller Messwerte für die Updatenachrichten sind, relativ zu ihren arithmetischen Mittelwerten gesehen und verglichen mit denen der Zyklusnachrichten, deutlich geringer. Dies spricht dafür, dass die Threads durch die insgesamt kürzeren Verarbeitungzeiten für Updatenachrichten seltener unterbrochen werden.

Rechenaufwand der Messrechner im Vergleich untereinander - Zyklusnachrichten Die zusammengefassten Ergebnisse der Messreihe 4 bezüglich des Zeitaufwandes der vier Messrechner zum Bearbeiten von Zyklusnachrichten sind in Abbildung 11.14 im Vergleich zueinander dargestellt. Zur besseren Erkennbarkeit der einzelnen Messabschnitte pro Messrechner sind diese auf der X-Achse wieder leicht versetzt angeordnet: Der jeweils erste Wert gehört zum Messabschnitt "Empfang bis Beginn Berechnung", der zweite zum Messabschnitt "Berechnung/Aktualisierung QuickLink-List" und der dritte Wert stellt die Summe beider Werte, also die Gesamtverarbeitungszeit, dar. Die genauen Werte zu diesem Diagramm sind in der zugehörigen Tabelle 11.13 aufgeführt.

Gut zu sehen ist, dass der Verarbeitsaufwand für die Nachrichten in allen Messabschnitten auf den verschiedenen Messrechnern unterschiedlich groß ist. Der Verlauf der Verarbeitungzeiten lässt sich mit den unterschiedlichen Rechenleistungen der Messrechner[7] erklären, die nicht unerheblich[8] sind. Die Verarbeitungszeit ist demnach, wie im Vorfeld erwartet wurde, von der Rechenleistung der Messrechner abhängig. Allerdings ist der Effekt nicht so stark ausgeprägt: Die beste durchschnittliche Verarbeitungsgesamtzeit hat nach Tabelle 11.13 der Messrechner *Thinkpad* mit 391,5 Mikrosekunden, dem gegenüber steht der Messrechner *Gericom* mit dem schlechtesten durchschnittlichen Wert von 876 Mikrosekunden, was relativ gesehen also etwa 224 % des besten Mittelwertes ausmacht. Bei den Maximalwerten, die hier von *Silentbob* mit 2144 Mikrosekunden und wiederum *Gericom* mit 3313 Mikrosekunden bestimmt werden, liegt der relative Unterschied nur bei 155 %. Die relativen Unterschiede wie auch die absoluten Verarbeitungszeiten lassen den Schluss zu, dass die Verarbeitungsaufwände für Zyklusnachrichten auf für weniger starke Rechner als zumutbar betrachtet werden können und die Ressourcen dieser nicht sonderlich in Anspruch nehmen.

[7] Vergleiche hierzu auch die technischen Daten der Messrechner in Tabelle 11.1 auf Seite 201.

[8] Auf die Rechenleistung der Messrechner wird in Abschnitt 11.4 eingegangen. Der Verlauf der Verarbeitungsaufwände unterhalb der Messrechner korrespondiert mit den ermittelten Rechenleistungen durch diverse Benchmarks und die eigenen Leistungsmessungen durch den PerformanceAnalyser.

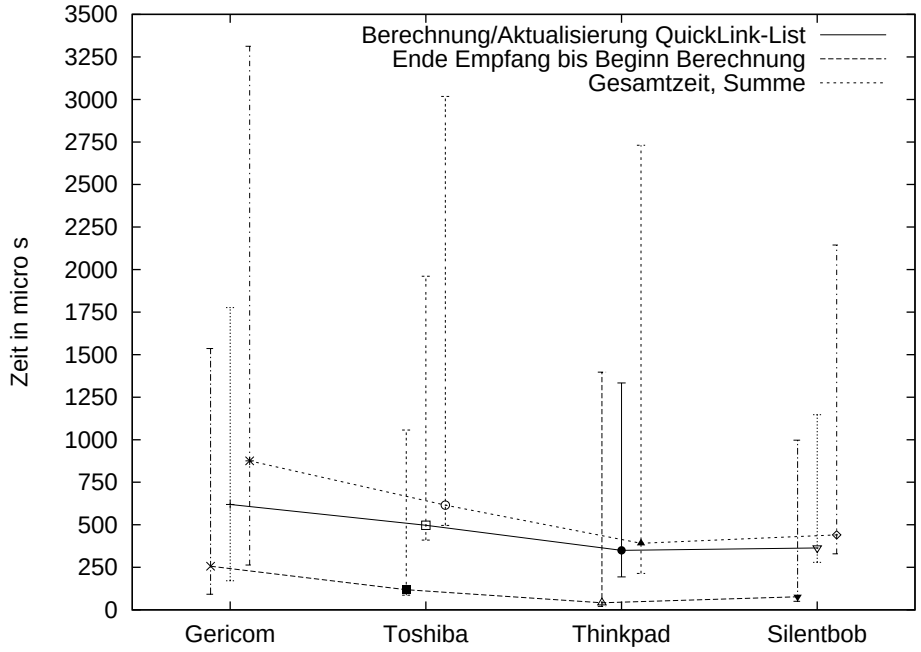

Abbildung 11.14: Die Verarbeitungzeiten von Zyklusnachrichten auf den Messrechnern in einem QuickLink-Netzwerk mit vier QuickLink-Knoten

Rechenaufwand der Messrechner im Vergleich untereinander - Updatenachrichten
Die zusammengefassten Ergebnisse der Messreihe 4 bezüglich des Zeitaufwandes der vier Messrechner zum Bearbeiten von Updatenachrichten sind in Abbildung 11.15 im Vergleich zueinander dargestellt. Die genauen Messwerte kann man der zugehörigen Tabelle 11.14 entnehmen.

Der Verlauf der Messwerte der Messrechner im Vergleich ähnelt der Betrachtung der Zyklusnachrichten in vorherigen Abschnitt und bestätigt den dort schon festgestellten unterschiedlich hohen zeitlichen Aufwand zum Verarbeiten von Nachrichten in Abhängigkeit von der Rechenleistung. Insgesamt gesehen liegt das Niveau der arithmetischen Mittelwerte, der Mediane und Maximalwerte für Updatenachrichten deutlich niedriger als bei den Zyklusnachrichten und kann mit der Hälfte derer abgeschätzt werden. Die durchschnittlichen zeitlichen Gesamtaufwände für die Verarbeitung der Updatenachrichten liegen bei den Messrechnern zwischen 120,6 Mikrosekunden bei *Thinkpad* und 543,2 Mikrosekunden bei *Gericom*, die Maximalwerte zwischen 248 Mikrosekunden bei *Thinkpad* und 1879 Mikrosekunden bei *Toshiba*. Der zeitliche Aufwand zur Verarbeitung von Updatenachrichten stellt also für schwächere Rechner ebenfalls keine Herausforderung dar.

Messrechner	zeitlicher Messabschnitt	Avg	Min	Max	Med
Gericom	Empfang bis Beginn Berechnung	256,6	92	1536	274
	Berechnung/Aktualisierung QuickLink-List	619,4	171,2	1777	589
	Gesamtzeit (Summe)	876	263,2	3313	863
Toshiba	Empfang bis Beginn Berechnung	118,3	86	1057	111
	Berechnung/Aktualisierung QuickLink-List	496,9	410	1961	452
	Gesamtzeit (Summe)	615,2	496	3018	563
Thinkpad	Empfang bis Beginn Berechnung	41,8	20	1397	28
	Berechnung/Aktualisierung QuickLink-List	349,7	194	1334	358
	Gesamtzeit (Summe)	391,5	214	2731	386
Silentbob	Empfang bis Beginn Berechnung	77	50	997	69
	Berechnung/Aktualisierung QuickLink-List	363,6	279	1147	360
	Gesamtzeit (Summe)	440,6	329	2144	429

Tabelle 11.13: Die Messwerte zur Verarbeitung von Zyklusnachrichten aller Messrechner im Vergleich in einem QuickLink-Netzwerk mit vier QuickLink-Knoten, wie sie im Diagramm in Abbildung 11.14 dargestellt sind; alle Angaben in Mikrosekunden

11.4 Praktische Evaluation des PerformanceAnalysers

Der PerformanceAnalyser in QuickLink soll die rechentechnische Performance des Hostcomputers QuickLinks messen. Für QuickLink ist aber nicht die absolute Performance des Hostrechners ausschlaggebend, sondern diejenige, die ihm über die Java VM, in der es ausgeführt wird, zur Verfügung gestellt wird. Daher misst der PerformanceAnalyser die Leistung nicht direkt auf dem Betriebssystem, sondern in der Java VM QuickLinks. Da im QuickLink-System ohnehin nur Vergleichwerte gebraucht werden, die eine grobe Klassifikation der Rechner erlauben, werden die gemessenen Performancewerte in ein diskretes Maß, die Priorität, umgerechnet, welches im Wertebereich des Datentyps *Byte* abgebildet wird und somit nur 256 unterschiedliche Performancestufen erlaubt.

Die Evaluation des PerformanceAnalysers sollte nun die Aussagekraft seiner

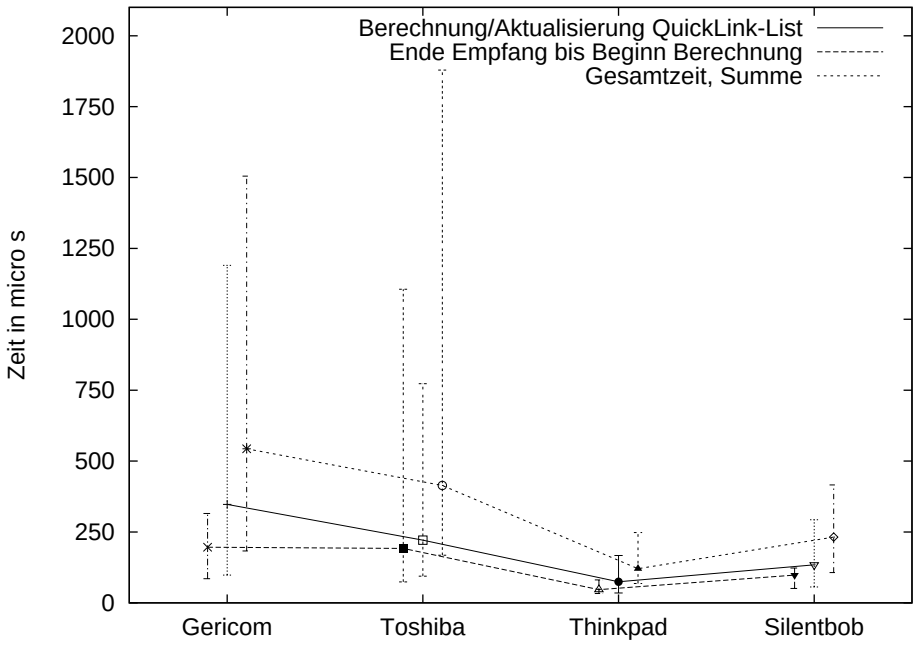

Abbildung 11.15: Die Verarbeitungzeiten von Updatenachrichten auf den Messrechnern in einem QuickLink-Netzwerk mit vier QuickLink-Knoten

Messwerte im Vergleich zu anderen, kurzlaufenden Leistungstestprogrammen und kleinen Benchmarks bestätigen. Dabei wurde nicht auf die absolute Präzision der Werte selbst und der Skalen der Messwerte Wert gelegt, sondern auf die relative Qualität der Leistungsaussagen: Ein starker Hostrechner sollte unter allen Leistungsmessungen auch als stark erscheinen, ein relativ schwacher Rechner entsprechend schwach. Dabei sollte vor allem der relative Leistungsunterschied zwischen den Messrechnern in allen Leistungstests möglichst gleich sein, also das "Profil" der Leistungswerte der Rechner übereinstimmen.

11.4.1 Messumgebungen und Messmethodik

Ein spezieller Messaufbau wie bei den Messungen im Netzwerk war für die Durchführung der Evaluation nicht notwendig, da die Leistungsmessungen und -tests auf jedem Messrechner isoliert durchgeführt wurden. Es wurden folgende Messrechner für die Leistungsmessungen verwendet:

Der PerformanceAnalyser wurde auf jedem Messrechner in jedem Betriebssystem mit einer Messreihe von 1000 Messwerten getestet. Alle Messungen wurden unter allen Betriebssystemen mit den Java VM SUN Java2, Version 1.5.0_10-b03, und bei Silentbob unter Linux mit der vergleichbaren SUN Java2, Ver-

Messrechner	zeitlicher Messabschnitt	Avg	Min	Max	Med
Gericom	Empfang bis Beginn Berechnung	196,2	85	315	201
	Berechnung/Aktualisierung QuickLink-List	347	98	1190	155
	Gesamtzeit (Summe)	543,2	183	1505	356
Toshiba	Empfang bis Beginn Berechnung	192,9	74	1106	87
	Berechnung/Aktualisierung QuickLink-List	221,1	94	773	195
	Gesamtzeit (Summe)	414	168	1879	282
Thinkpad	Empfang bis Beginn Berechnung	46,5	33	81	40
	Berechnung/Aktualisierung QuickLink-List	74,1	35	167	67,5
	Gesamtzeit (Summe)	120,6	68	248	107,5
Silentbob	Empfang bis Beginn Berechnung	97,8	51	123	108,5
	Berechnung/Aktualisierung QuickLink-List	134,2	56	293	127
	Gesamtzeit (Summe)	232	107	416	235,5

Tabelle 11.14: Die Messwerte zur Verarbeitung von Updatenachrichten aller Messrechner im Vergleich in einem QuickLink-Netzwerk mit vier QuickLink-Knoten, wie sie im Diagramm in Abbildung 11.15 dargestellt sind; alle Angaben in Mikrosekunden

sion 1.5.0_10-b03 64-Bit Server VM, durchgeführt. Als Vergleichsmaßstab für die gemessenen Prioritätswerte wurden auf jedem Messrechner Messungen mit je vier kleinen Benchmarkprogrammen unter dem jeweiligen Windows-Betriebssystem durchgeführt. Die Benchmarks wurden jeweils zehnmal ausgeführt und dann der arithmetische Mittelwert der Ergebnisse weiterverwendet. Die Benchmarks kamen mit den folgenden Parametern zum Einsatz:

- CPUBench2003 [Fra07], Version 1.5 Beta 2
  - ohne die Optionen *Software Raytracing* und
  - ohne *Direct3D Software Rendering*, da diese beiden die Leistung des grafischen Systems des Rechners mit einbeziehen, welche aber hier nicht relevant ist und daher die Gesamtleistung verfälschen würde

Name	Prozessortyp	MHz	RAM	Speicher-typ	Betriebssysteme
Armada	Intel Pentium II	266	192 MB	SD	Linux 2.4.19 / Win98 SE
Toshiba	Intel Pentium IV	1600	256 MB	SD	Linux 2.6.18 / WinXP Home
Gericom	Intel Pentium IV	2000	448 MB	SD	Linux 2.6.16 / WinXP Prof. SP2
Thinkpad	Intel T2500	2000	2048 MB	DDR2	Linux 2.6.16 / WinXP Prof. SP2
Silentbob	AMD Athlon64 3200+	2000	1024 MB	DDR1	Linux 2.6.16 (64Bit) / WinXP Prof. SP2
IPC677	Intel Pentium 4	2800	1024 MB	DDR1	Linux 2.6.16 / WinXP Prof. SP2

Tabelle 11.15: Die für die Leistungsmessungen verwendeten Messrechner mit einigen technischen Daten

- MCS CPU Benchmark V5 [Kwi07], Version 5.03
  - ohne Einschränkungen
- Hexus PiFast [Xav07], Version 4.1, für kurze Laufzeiten:
  - Algorithmus: Chudnovsky-Methode
  - Berechnungsmethode: Standard Mode (ohne Festplattencache)
  - Anzahl der Dezimalstellen (digits): 1 Million
  - FFT Size: 128 k; Compressed output: yes
- Super Pi [KY07], Version 1.5, für kurze Laufzeiten:
  - Anzahl der Dezimalstellen (digits): 1 Million.

Der *MCS CPU Benchmark* liefert als Ergebnis eine Vergleichszahl, welche mit keiner Einheit spezifiziert ist. Sie drückt in ihrer Höhe die Leistung des Rechners aus und kann daher direkt zum Vergleich der Rechner untereinander herangezogen werden. Die Ergebnisse des *CPUBench2003*, des *Hexus PiFast* und des *Super Pi* liefern hingegen die Zeit zurück, welche ein Rechner braucht, um den jeweiligen Benchmark erfolgreich durchzuführen. Da hier also ein kleinerer Wert Ausdruck einer besseren Leistung ist, wurden für die Diagramme in der Auswertung die Kehrwerte der Messzeiten gebildet, um eine mit der *Prioritätsmessung*

des PerformanceAnalysers und dem *MCS CPU Benchmark* vergleichbare Leistungszahl (große Leistung = große Zahl) zu erhalten.

11.4.2 Ergebnisse und Diskussion

Die Messergebnisse der Leistungstests wurden in jeweils einem Diagramm, welches die Leistungswerte der Messrechner bezüglich eines Leistungstests vergleichbar gegenüberstellt, in Abbildung 11.16 dargestellt. Aus den Diagrammen lassen sich gut die Verhältnisse der Leistungen der Messrechner zueinander ablesen. Die Reihenfolge der dargestellten Messrechner wurde in jedem Diagramm beibehalten, um eine leichtere Vergleichbarkeit der Diagramme und damit der Leistungstests untereinander zu gewährleisten. In jedem Diagramm sind auf der X-Achse die Messrechner als Kategorie und auf der Y-Achse die jeweilige Leistungskennzahl dargestellt. Man beachte, dass die Leistungstests *Super Pi* und *CPUBench2003* auf dem Messrechner *Armada* wegen seiner geringen Ressourcen nicht ausgeführt werden konnten und daher in den betreffenden Diagrammen kein Wert angegeben ist. Der Messrechner ist dort auch zusätzlich mit einem kleinen *(n)* für *nicht ausführbar* gekennzeichnet. Alle in den Diagrammen abgebildeten Messwerte können ebenfalls aus der zugehörigen Tabelle 11.16 entnommen werden.

Im Vergleich der verschiedenen Leistungstests und der Prioritätswerte des PerformanceAnalysers untereinander fallen die Messrechner *Silentbob* und *Thinkpad* als die leistungsstärksten auf, was auch deren realer Ausstattung mit Rechenleistung und Speicherplatz entspricht. Der Messrechner *Armada* bildet bei allen Leistungstests, an welchem er teilnehmen konnte, das Schlusslicht. Auch dies war, im Hinblick auf die Ausstattung der Messrechner im Vergleich, zu erwarten. Die Messrechner *IPC677*, *Toshiba* und *Gericom* teilen sich leistungsmäßig das Mittelfeld. Die "Profile" der nebeneinander gestellten Säulen der Diagramme ergeben, ausgenommen der Prioritätsmessung unter Linux, durchaus ähnliche Muster: Die leistungsmäßige Reihenfolge der Rechner ist demnach aufsteigend *Armada*, *Toshiba*, *Gericom* und *IPC677*, danach folgt die Spitzengruppe, in der sich *Silentbob* und *Thinkpad*, je nach Leistungstest, in der Führung abwechseln.

Die Abweichungen der gemessenen Leistungsniveaus einzelner Messrechner voneinander in einzelnen Leistungstests ist auf die unterschiedlichen Tests selbst zurückzuführen. Jeder Leistungstest oder Benchmark verfolgt seine eigene Strategie, einen möglichst aussagekräftigen Vergleichswert zu liefern. Insofern werden verschiedene Belastungsarten zu einem Test zusammengesetzt, wobei die Zusammensetzung der einzelnen Belastungsarten und auch deren Intensität oder Betonung in allen Leistungstests unterschiedlich ausfällt. Jeder Messrechner schneidet daher in den Tests unterschiedlich gut ab, je nachdem, ob er seine Stärken gegenüber anderen Rechnern im Leistungstest ausspielen kann oder nicht. Die Gesamttendenz bleibt aber dennoch gleich: Ein starker Rechner bleibt, über alle

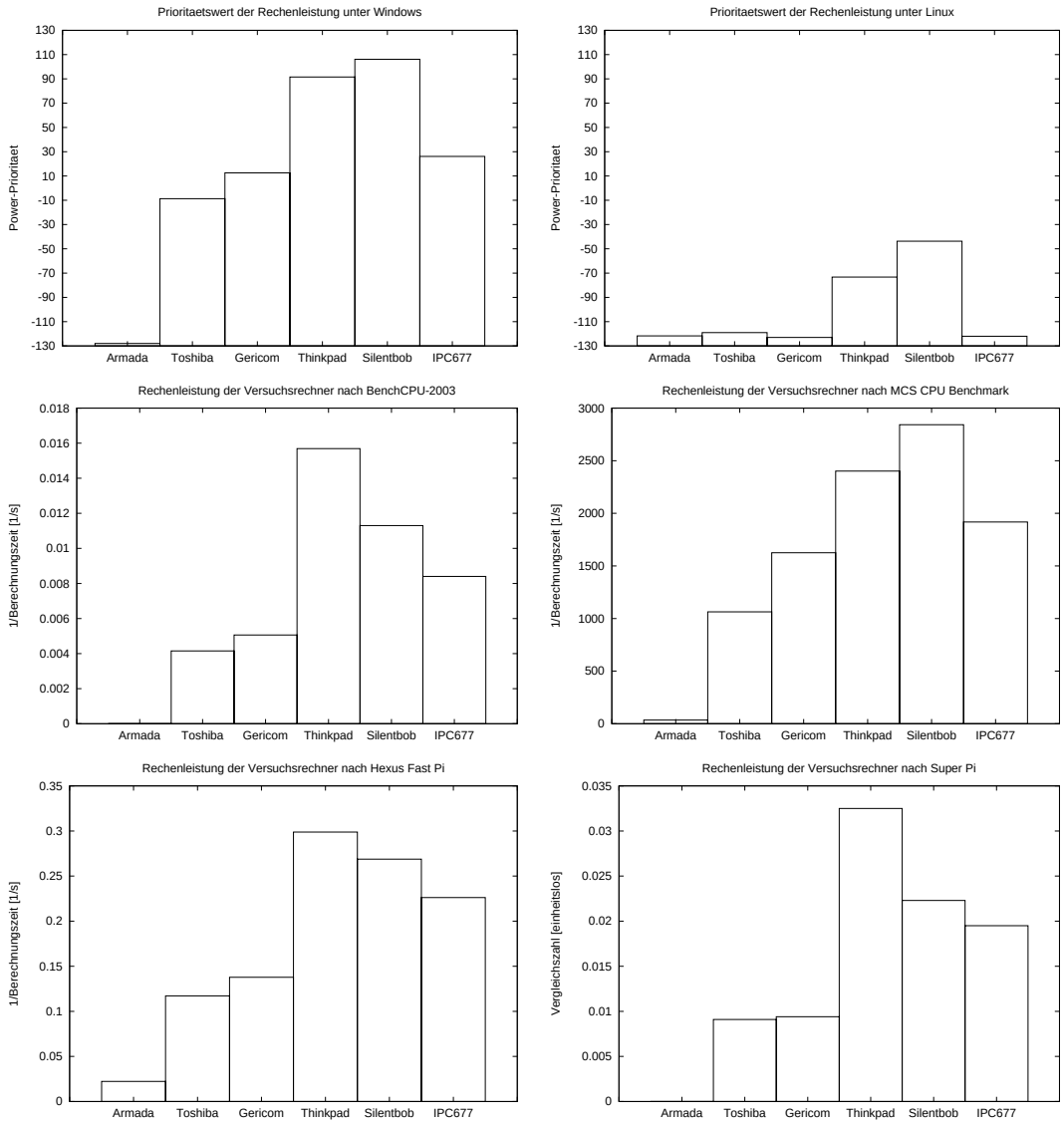

Abbildung 11.16: Die gemessenen Leistungskennzahlen der Messrechner mit allen Leistungstests im Vergleich, die oberen beiden Diagramme zeigen die Prioritätsmessungen mit dem PerformanceAnalyser; bei *CPU-Bench2003* und *Super Pi* konnten keine Werte für den Messrechner *Armada* ermittelt werden

Tests hinweg gesehen, stark und umgekehrt.

Eine Abweichung von der Gesamtcharakteristik im Leistungstest bildet die

Leistungsmessung mit dem PerformanceAnalyser unter Linux. Auch hier können sich zwar die Messrechner *Silentbob* und *Thinkpad* als eindeutige Spitzengruppe etablieren, allerdings sind deren relative Abweichungen zu den Leistungswerten der anderen Messrechner deutlich größer, so dass sich nicht das "Profil" der Säulen einstellt. Auch die relativen Messwerte der Rechner *Armada*, *Toshiba*, *Gericom* und *IPC677* entsprechen nicht den erwarten Verhältnissen aus den anderen Leistungstests. Ferner zeigt der Vergleich der absoluten PerformanceAnalyser-Messwerte zwischen den verschiedenen Betriebssystemen Windows und Linux, dass hier erhebliche Abweichungen existieren. Die Messwerte sind, da sie auf der gleichen Skala beruhen, direkt miteinander vergleichbar, weichen aber stark ab. Dies lässt sich nur auf die doch recht unterschiedlichen Implementierungen der verwendeten SUN JAVA VMs für beide Betriebssysteme zurückführen, welche zu einem unterschiedlichen Verhalten bezüglich der Leistungsmessung führen.

Leistungs-test		Armada	Toshiba	Gericom	Thinkpad	Silentbob	IPC677
Perf.Analy.	Win	-128	-8,71	12,56	91,55	106,14	26,12
	Linux	-121,70	-119,01	-123,01	-73,25	-43,74	-122
CPUBench 2003	Zeit [s]	x	241,12	197,72	63,75	88,46	119,05
	1/Zeit [1/s]	x	0,0041	0,0051	0,0157	0,0113	0,0084
MCS CPU Benchm.		34	1063	1625	2402	2842	1917
Hexus PiFast	Zeit [s]	44,8	8,54	7,26	3,35	3,72	4,42
	1/Zeit [1/s]	0,0233	0,1171	0,1378	0,2988	0,2688	0,2262
Super Pi	Zeit [s]	x	109,8	106,2	30,7	44,9	51,4
	1/Zeit [1/s]	x	0,0091	0,0094	0,0325	0,0223	0,0195

Tabelle 11.16: Die Ergebnisse der verschiedenen Leistungsmessungen

Insgesamt kann die Qualität der Leistungsmessung mit Hilfe des Performance-Analyser als ausreichend angesehen werden, um im QuickLink-Netzwerk vorhandene Knoten in ihrer potentiellen Leistungsfähigkeit untereinander vergleichen und einordnen zu können.

11.5 Zusammenfassung

Die Ergebnisse der Evaluation des Prototyps QuickLink zeigen, dass ein lokales logisches Netzwerk sehr wohl eine hohe Netzwerkdynamik abbilden kann und dabei keine große Netzwerkbelastung erzeugen muss. Dabei erfolgt der Eintritt neuer QuickLink-Knoten in das logische QuickLink-Netzwerk typischerweise innerhalb weniger Sekunden. Auch unter schlechten Netzwerkbedingungen kann das logische Netzwerk, ohne selbst große Last zu erzeugen, zusammengehalten werden und unterstützt so verschiedenste dynamische Effekte, wie sie in heutigen Netzwerken anzutreffen sind. Durch die geringen Ressourcenanforderungen QuickLinks wird es auch schwächeren Hostcomputern ermöglicht, an verteilten Anwendungen teilhaben zu können.

Die Leistungsbewertung der Ausführungsumgebungen für verteilte Anwendungen auf den Hostcomputern ermöglicht eine genauere Einschätzung der leistungsbezogenen Möglichkeiten eines QuickLink-Knotens im logischen Netzwerk. Dadurch wird einerseits die gleichwertige Betrachtung von Peers in verteilten Systemen erweitert, indem die Transparenz bezüglich leistungsbezogener Eigenschaften der hostenden Plattformen aufgehoben wird. Andererseits schafft dies die Grundlagen für eine dynamische Erweiterung der Netzwerkstruktur um Peers, welche zusätzliche Rollen annehmen können und mehr Verantwortung übernehmen sollen (Regions-Manager).

12 ServiceJuggler

Dem Infrastrukturdienst ServiceJuggler obliegt innerhalb QuickLinkNets die zuverlässige Verwaltung der Anwendungsdienstinformationen und die Steuerung der Rollenverteilung der QuickLink-Knoten innerhalb einer Region. Dazu verwenden die Komponenten ServiceJugglers verschiedene Mechanismen, die es ihnen ermöglichen, den entscheidenden funktionalen Teil, nämlich das Verzeichnis für Anwendungsdienste, von einem zum anderen QuickLink-Knoten migrieren zu lassen. Alle Komponenten und deren Funktionen der prototypischen Implementierung ServiceJugglers wurden auf ihre Funktionalität hin praktisch getestet. Dabei wurde vor allem auf die dynamischen Aspekte, wie die Wahl des Regions-Managers und die Übertragung der ServiceRegistry, Wert gelegt.

Das zuverlässige Funktionieren einer Region in dynamischen Situationen kann nur dann erfüllt werden, wenn der Dienst zur Verwaltung der Anwendungsdienstinformationen *ServiceRegistry*, und damit die Anwendungsdienstinformationen selbst, nach einer Migration schnell wieder zur Verfügung gestellt werden kann. Daher stehen die mit dem *ServiceRegistry* zusammenhängenden Leistungsaspekte ServiceJugglers im Mittelpunkt seiner Evaluation.

12.1 Messumgebungen

Die Messungen wurden in einem Ethernet nach IEEE 802.3u mit einer Bandbreite von 100 MBit/s und einem Wireless LAN nach IEEE 802.11b mit 11 MBit/s Bandbreite durchgeführt.

Für jeden Netzwerktyp wurde, wie in Abbildung 12.1 skizziert, jeweils eine Messumgebung aufgebaut. Die dritte Messumgebung stellt eine Kombination aus Ethernet und Wireless LAN dar und simuliert eine in der Realität typisch vorkommende Infrastruktur, bei der ein Ethernet über einen Wireless LAN Access Point erweitert wird. Als Messhardware dienten wieder die schon aus der Evaluation QuickLinks in Kapitel 11 bekannten Geräte, die in den Tabellen 11.1 und 11.2 auf Seite 201 nachgelesen werden können. Sie wurden je nach Bedarf, wie in Abbildung 12.1 zu sehen, zur betreffenden Messumgebung kombiniert.

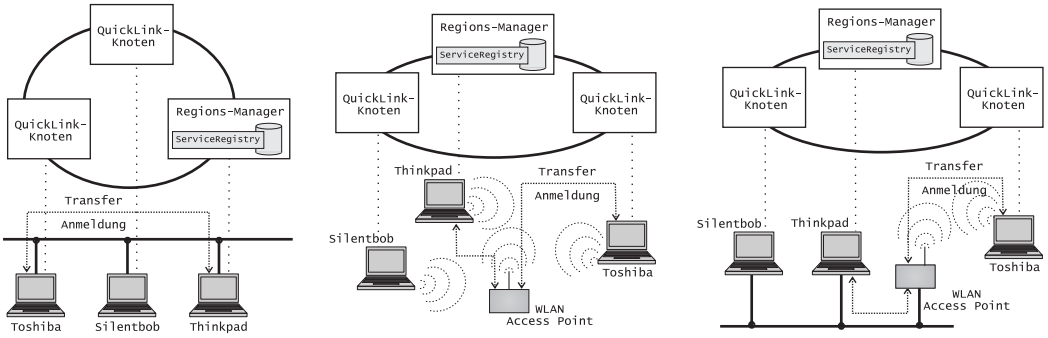

Abbildung 12.1: Die Messumgebungen zur Evaluation ServiceJugglers: Links die Messumgebung im Ethernet, in der Mitte die Messumgebung im Wireless LAN und rechts die aus Ethernet und Wireless LAN kombinierte Messumgebung

12.2 Migration der Daten eines ServiceRegistry

12.2.1 Messmethodik

Ziel der Messungen war es, den zeitlichen Aufwand zur Migration eines ServiceRegistry beim Wechsel der Regions-Manager-Rolle von einem Knoten auf einen anderen zu bestimmen.

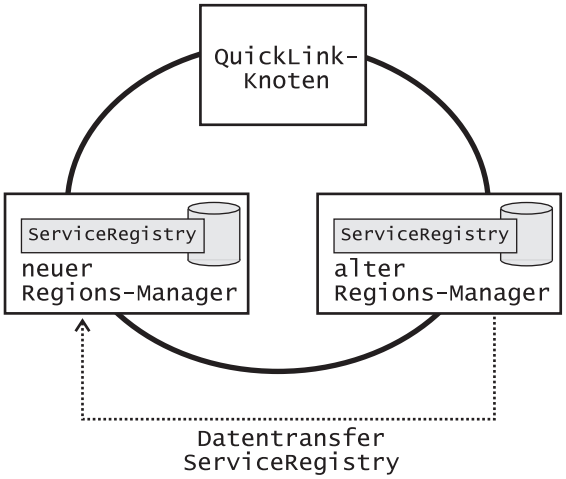

Abbildung 12.2: Das Schema der Datenübertragung der Messreihen zur Datenmigration von einem ServiceRegistry zum anderen

Die dazu erforderliche Zeit wird von der Übertragungszeit der Daten des ServiceRegistry vom Regions-Manager zum anderen QuickLink-Knoten, der die

Rolle des Regions-Managers übernehmen will, dominiert. Die Übertragungszeit ist von der Größe der Datenbasis und der Leistungsfähigkeit des unterliegenden Netzwerkes abhängig. Daher wurden Messexperimente mit unterschiedlichen Netzwerkstrukturen, wie sie im vorherigen Abschnitt 12.1 in Form von Messumgebungen vorgestellt wurden, durchgeführt. In jedem Messexperiment wurden Messreihen mit unterschiedlichen ServiceRegistry-Größen, ausgedrückt durch die Anzahl der registrierten Anwendungsdienste, realisiert. Das Schema der Datenübertragung einer jeden Messreihe im logischen Netzwerk ist in Abbildung 12.2 dargestellt.

12.2.2 Ergebnisse Ethernet

Eine Aufstellung der Messexperimente und Messreihen der Migration der ServiceRegistry-Daten im Ethernet sowie deren Ergebnisse enthält Tabelle 12.1. Die Ergebnisse in grafischer Form zeigt das zugehörige Diagramm in Abbildung 12.3. Dort sind die arithmetischen Mittelwerte der Migrationszeiten über der Anzahl der zu migrierenden Dienste aufgetragen. Die Errorbars zeigen die Minimum- und Maximumwerte der jeweiligen Messreihe an.

Anzahl Dienste	Ethernet 100 MBit		
	Avg	Min	Max
100	0,62	0,42	0,71
1000	0,84	0,72	0,88
5000	2,09	2,02	2,20
7000	4,92	4,87	4,99
10000	14,1	13,9	14,2
12000	24,2	23,8	25,4
14000	38,0	37,9	38,1
16000	56,0	55,8	56,1
18000	81,5	80,6	82,4
20000	113,9	113,3	114,6

Tabelle 12.1: Die Messergebnisse in Sekunden zur Übertragung der Daten des ServiceRegistry im Ethernet in Abhängigkeit von der Anzahl der registrierten Dienste, wie sie im zugehörigen Diagramm 12.3 dargestellt sind

Die Y-Achse des Diagramms ist logarithmisch geteilt, um den großen Zeitbereich besser abbilden zu können. Wie man aus dem Verlauf der Zeiten zur Migration entnehmen kann, ist die Abhängigkeit der Übertragungzeit von der Anzahl der zu übertragenden Dienste schlechter als linear. Da während der Mes-

sungen die Messumgebung isoliert blieb und die Bandbreite nicht variiert wurde, war die Datenübertragungsrate in der Messumgebung konstant groß. Die nichtlineare Migrationzeit beruht auf einer nichtlinearen Abhängigkeit der Größe des ServiceRegistry von der Anzahl der registrierten Dienste, welche auf die Art der prototypischen Implementierung des ServiceRegistry zurückzuführen ist.

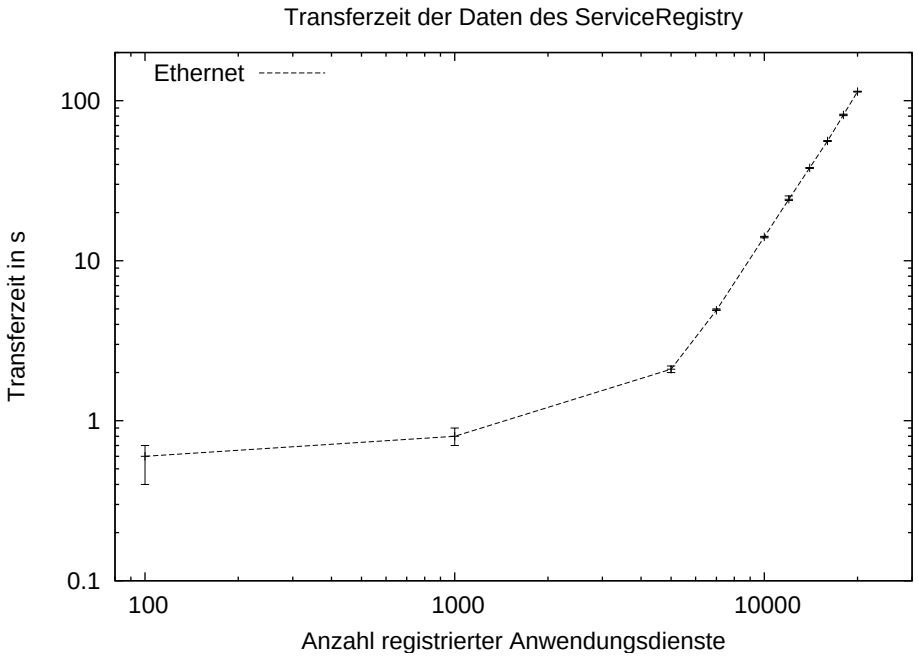

Abbildung 12.3: Die Migrationszeit eines ServiceRegistry im Ethernet in Abhängigkeit von der Anzahl seiner registrierten Dienste

Die Ergebnisse zeigen, dass die Übertragungzeit der Daten eines ServiceRegistry im 100 MBit/s Ethernet bis zu einer Anzahl von 5000 registrierten Diensten maximal etwa 2 Sekunden beträgt, also unter den für QuickLink angenommenen Zeitschranken der abbildbaren Netzwerkdynamik als sehr performant gelten kann.

Die Anzahl von 5000 Anwendungsdiensten stellt nach unserer Einschätzung sicherlich eine Höchstgrenze für eine Region dar, da in QuickLink nur maximal 127[1] Knoten verwaltet werden können. Unter diesen Umständen ist, bei durchschnittlich maximal 10 Anwendungsdiensten pro QuickLink-Knoten als realistische Annahme, mit einer wahrscheinlichen Anzahl von etwa maximal 1300 Anwendungsdiensten in einer Region zu rechnen. Ein ServiceRegistry mit dieser

[1] In der aktuellen prototypischen Implementierung ist die Anzahl der maximal verwaltbaren QuickLink-Knoten, wie auf Seite 196 dargestellt, auf 127 beschränkt.

Anzahl von registrierten Anwendungsdiensten ist dann im Ethernet in einer Zeit von unter 2 Sekunden übertragbar.

12.2.3 Ergebnisse Wireless LAN

Im Wireless LAN wurden die gleichen Messexperimente wie im Ethernet durchgeführt. Allerdings wurden hier unterschiedliche Netzwerkkonstellationen und unterschiedliche Übertragungsoptionen verwendet, um das Verhalten eines Wireless LAN auf die Übertragungszeiten zu untersuchen. Die Messexperimente und Messreihen sowie deren Ergebnisse sind in Tabelle 12.2 festgehalten und im Diagramm in Abbildung 12.4 grafisch dargestellt.

Anzahl Dienste	Ethernet-WLAN unverschlüsselt			Ethernet-WLAN WEP 128 Bit			WLAN-WLAN WEP 128 Bit		
	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max
100	7,9	6,2	8,9	15,6	12,3	17,6	26,0	23,3	26,9
1000	10,5	9,3	10,9	20,7	18,4	21,6	35,9	31,9	37,8
5000	26,5	26,0	26,8	52,5	51,6	52,6	90,9	89,3	91,0
7000	61,6	62,3	63,5	123	121	125	214	212	217
10000	179	178	181	354	351	356	613	610	615
12000	306	306	320	605	604	608	1050	1048	1072
14000	472	471	474	933	931	940	1626	1621	1634
16000	714	713	716	1404	1399	1407	2427	2440	2447
18000	1033	1031	1039	2036	2030	2054	3531	3516	3543
20000	1445	1443	1451	2853	2851	2856	4979	4958	4983

Tabelle 12.2: Die Messergebnisse in Sekunden zur Übertragung der Daten des ServiceRegistry in verschiedenen, WLAN-basierten Netzwerkumgebungen in Abhängigkeit von der Anzahl der registrierten Dienste, wie sie im zugehörigen Diagramm 12.4 dargestellt sind

Im Diagramm in Abbildung 12.4 stellen die Messpunkte jeweils das Ergebnis einer Messreihe in Form ihres arithmetischen Mittelwertes dar. Die Errorbars bezeichnen die während der jeweiligen Messreihe aufgetretenen Maximum- und Minimumwerte. Wie bei den Messungen im Ethernet ist bei allen drei Messexperimenten wieder die nichtlineare Abhängigkeit der Übertragungszeiten von der Anzahl der registrierten Dienste des ServiceRegistry zu erkennen. Die Übertragungszeiten im unverschlüsselten Wireless LAN liegen insgesamt über eine Zehnerpotenz höher als im Ethernet, was sich leicht aus den nominalen Bitübertragungsraten von 100 MBit/s im Ethernet und von 11 MBit/s im verwendeten Wireless LAN erklärt. Ferner wirkt sich hier die im Zugriffsverfahren CSMA/CA

von Wireless LAN begründete Reduzierung der Rohbitdatenrate aus, so dass nur ein geringerer Wert als 11 MBit/s für die reale Datenübertragung [Wik07f] zur Verfügung steht.

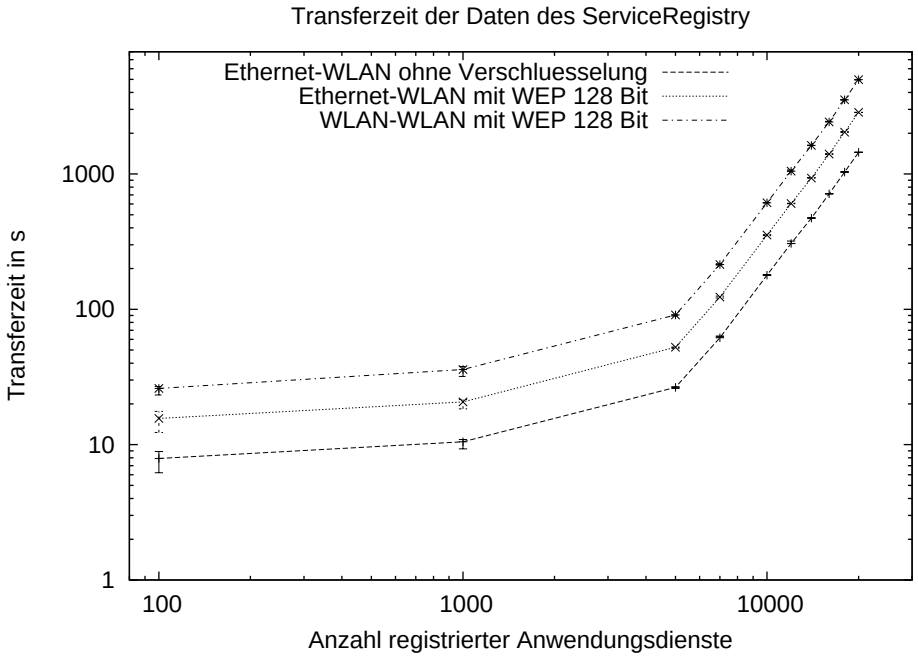

Abbildung 12.4: Die Migrationszeiten eines ServiceRegistry in Abhängigkeit von der Anzahl seiner registrierten Dienste; Messexperimente in verschiedenen, Wireless LAN-basierten Netzwerkkonstellationen

Die Messwerte zwischen den verschiedenen Wireless LAN-basierten Messumgebungen weichen z.T. erheblich voneinander ab. So verdoppeln sich in etwa die Messwerte zwischen den Messreihen "Ethernet-WLAN ohne Verschlüsselung" und "Ethernet-WLAN mit WEP 128 Bit", was auf den zusätzlichen Aufwand für die Verschlüsselung der Daten zurückzuführen ist. Vom Messexperiment "Ethernet-WLAN mit WEP 128 Bit" zu den Messreihen des Messexperimentes "WLAN-WLAN mit WEP 128 Bit" verdoppeln sich die Werte fast noch einmal. Dies liegt am geteilten Medium der Luftschnittstelle des Wireless LAN: Da der Datenstrom zuerst vom Sender *Thinkpad* an den Access Point und von diesem an den Empfänger *Toshiba* gesendet wird, ist der Datenstrom auf dem Medium zweimal unterwegs. Die beiden Datenströme müssen sich daher das Medium teilen, woraus sich die Verringerung der Datenübertragungsrate ergibt.

Bei der Untersuchung der Übertragungszeit der ServiceRegistry im Ethernet im Abschnitt 12.2.2 wurde eine realistische Einschätzung von ca. 1300 registrier-

ten Anwendungsdiensten in einem QuickLink-Netzwerk von maximal 127 Knoten getroffen. Unter diesen Bedingungen liegt die Übertragungszeit einer ServiceRegistry (mit 1300 Diensten), aus dem Diagramm in Abbildung 12.4 abgelesen, bei etwa 10 Sekunden im unverschlüsselten Netzwerk. Da ein unverschlüsseltes Wireless LAN in der Praxis nicht sinnvoll betreibbar ist, ist dieser Wert in der Praxis nicht relevant. Vielmehr ist der Wert für die Kombination "Ethernet-WLAN mit WEP 128 Bit" realistisch, hier liegt die Übertragungszeit laut Diagramm bei etwa 40 Sekunden. Im reinen verschlüsselten Wireless LAN liegt die Übertragungszeit dann, abgelesen aus dem Diagramm in Abbildung 12.4, bei etwa 70 Sekunden.

Ausgehend von diesen Werten zeigt sich, dass die Migration einer großen ServiceRegistry auf einen QuickLink-Knoten, der über ein Wireless LAN 802.11b angeschlossen ist, möglich ist, aber einen erheblichen Performancenachteil darstellen kann. Ein ServiceRegistry sollte daher nur auf einem Rechner mit einer starken Netzwerkanbindung betrieben werden. Dies korrespondiert aber mit der realistischen Annahme, dass sich logische Netzwerke im Internet meist über eine heterogene Netzwerk-Infrastruktur erstrecken, welches aus einem Kernnetzwerk aus fest angeschlossenen und gut vernetzten Rechnern mit hoher Verfügbarkeit bestehen und die über Funknetzwerke an der Peripherie erweitert werden, um mobile Rechner (meist nur zeitweise) einbinden zu können. Zentrale Dienste, wie der ServiceRegistry, sollen nur auf geeigneten, also potenten Rechnern betrieben werden, weshalb nur ein solcher zum Regions-Manager gewählt werden darf.

12.3 Anmeldezeiten am ServiceRegistry

12.3.1 Messmethodik

Unterstellt man im QuickLink-Netzwerk eine Dynamik durch ein- und austretende QuickLink-Knoten, so kommt es zwangläufig zu Anmeldungen von Anwendungsdiensten am ServiceRegistry. Je höher die Netzwerkdynamik ist, desto öfter werden auch Anwendungsdienste registriert werden. Ziel der Messungen war es daher, den zeitlichen Aufwand eines QuickLink-Knotens zur Anmeldung eines Anwendungsdienstes herauszufinden. Die Anmeldezeit ist zum einen von der Leistungsfähigkeit des unterliegenden Netzwerkes und zum anderen von der Verarbeitungsgeschwindigkeit des ServiceRegistry abhängig. Daher wurden die Messungen in den in Abschnitt 12.1 beschriebenen Messumgebungen durchgeführt, um den Einfluss der Netzwerkinfrastruktur auf die Anmeldezeiten herauszufinden. In jedem Messexperiment wurden Messreihen mit einer unterschiedlichen Anzahl von schon registrierten Diensten durchgeführt, um den Einfluss der ServiceRegistry-Größe auf die Anmeldezeit zu erfahren.

Die Messungen selbst liefen nach dem in Abbildung 12.5 skizzierten Schema

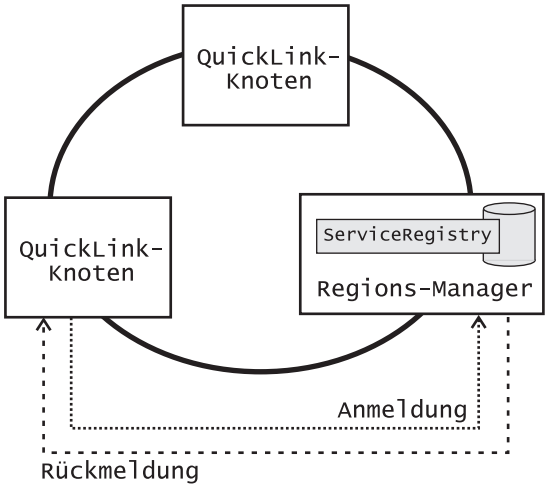

Abbildung 12.5: Das Schema der übermittelten Nachrichten, die zur Messung der Anmeldezeiten am ServiceRegistry genutzt wurden

ab: Ein QuickLink-Knoten versucht, einen Dienst am ServiceRegistry anzumelden. Er misst dabei die Zeit vom Beginn der Anmeldung bis zum Eintreffen der Antwort vom ServiceRegistry, dass der Dienst erfolgreich angemeldet wurde.

12.3.2 Ergebnisse Ethernet

Tabelle 12.3 zeigt die Ergebnisse der aufgenommenen Messreihen des Messexperimentes im Ethernet. Die Ergebnisse sind als arithmetisches Mittel der Messreihen mit ihren Minimum- und Maximumwerten aufgeführt. Abbildung 12.6 illustriert die Messergebnisse in Form eines Diagramms, bei dem die arithmetischen Mittel der Anmeldezeiten über der Größe des ServiceRegistry aufgezeigt sind. Die Errorbars stellen die jeweiligen Minimum- und Maximumwerte dar.

Die Ergebnisse zeigen, dass die Abhängigkeit der Anmeldezeit von der Anzahl der schon registrierten Dienste schlechter als linear ist. Allerdings ist die Ausprägung nicht so stark wie bei der Migration einer ServiceRegistry. Auch die absoluten Werte sind im Vergleich zur Migration recht gering, sie liegen bei einem ServiceRegistry mit 1000 Diensten bei durchschnittlich 300 ms und steigern sich bei einer 10000 Dienste großen ServiceRegistry auf durchschnittlich 3,77 Sekunden. Da die Netzwerkeigenschaften und die Nachrichtengröße zur Anmeldung während der Messungen nicht geändert wurden und folglich konstant sind, ist die Steigerung der Anmeldezeiten allein auf die gestiegene Verarbeitungszeit des ServiceRegistry zurückzuführen.

Bezogen auf eine realistische ServiceRegistry-Größe von 1300 registrierten Anwendungsdiensten sollte die durchschnittliche Anmeldezeit unter 340 ms liegen,

Anzahl Dienste	Ethernet 100 MBit		
	Avg	Min	Max
1000	0,30	0,27	0,34
2000	0,32	0,28	0,36
4000	0,65	0,59	0,69
6000	1,43	1,36	1,47
8000	2,53	2,45	2,69
10000	3,77	3,73	3,81

Tabelle 12.3: Die Messergebnisse in Sekunden zum Anmelden eines Dienstes am ServiceRegistry im Ethernet in Abhängigkeit von der Anzahl der registrierten Dienste, wie sie im zugehörigen Diagramm 12.6 dargestellt sind

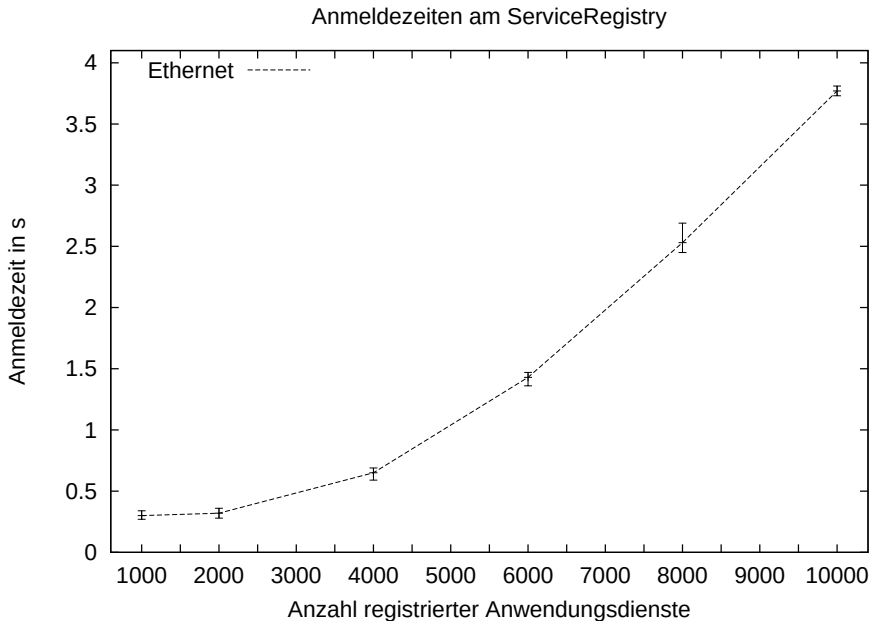

Abbildung 12.6: Die Anmeldezeit eines Anwendungsdienstes am ServiceRegistry im Ethernet in Abhängigkeit von der Anzahl seiner registrierten Dienste

was die Anmeldezeit bei 2000 registrierten Diensten darstellt. Die Anmeldezeit im Ethernet ist daher, im Vergleich zur Eintrittszeit eines QuickLink-Knotens, die nicht unter einer Zykluszeit (empfohlener Wert 1 s) liegen kann, als sehr kurz einzustufen und unterstützt die geforderte Dynamik im QuickLink-Netzwerk sehr gut.

12.3.3 Ergebnisse Wireless LAN

Die Ergebnisse der Messexperimente in den verschiedenen Wireless LAN-basierten Messumgebungen sind in Tabelle 12.4 in Form der arithmetischen Mittelwerte, der Minima und der Maxima aller Messreihen aufgeführt.

Anzahl Dienste	Ethernet-WLAN unverschlüsselt			Ethernet-WLAN WEP 128 Bit			WLAN-WLAN WEP 128 Bit		
	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max
1000	0,32	0,28	0,36	0,34	0,29	0,38	0,38	0,33	0,42
2000	0,34	0,29	0,37	0,37	0,32	0,41	0,41	0,34	0,44
4000	0,67	0,59	0,69	0,70	0,61	0,71	0,74	0,64	0,76
6000	1,43	1,38	1,46	1,42	1,37	1,48	1,52	1,41	1,56
8000	2,55	2,42	2,65	2,58	2,46	2,70	2,62	2,48	2,72
10000	3,78	3,71	3,79	3,81	3,62	3,70	3,86	3,78	3,99

Tabelle 12.4: Die Messergebnisse in Sekunden zum Anmelden eines Dienstes am ServiceRegistry in verschiedenen, WLAN-basierten Netzwerkumgebungen in Abhängigkeit von der Anzahl der registrierten Dienste, wie sie im zugehörigen Diagramm 12.7 dargestellt sind

In Abbildung 12.7 sind die Messwerte in einem Diagramm grafisch aufbereitet dargestellt. Die Werte zeigen das gleiche Verhalten wie die Messwerte im Ethernet, also eine schlechter als lineare Abhängigkeit der Anmeldezeiten von der Anzahl der schon registrierten Anwendungsdienste. Auch die absolute Höhe der Messwerte lässt sich mit den Werten im Ethernet vergleichen: So liegt die niedrigste Anmeldezeit bei 1000 Diensten in der Ethernet-WLAN-Kombination mit unverschlüsselter Kommunikation bei 320 ms, die höchste Anmeldezeit im reinen, verschlüsselt sendenden Wireless LAN bei 10000 angemeldeten Diensten hingegen bei durchschnittlich 3,86 Sekunden. Hier zeigt sich, dass die geringe zu übertragende Datenmenge beim Anmelden eines Dienstes trotz der unterschiedlichen Leistungspotentiale der verschiedenen Messumgebungen keinen bestimmenden Einfluss auf die Gesamtanmeldezeit hat. Diese wird vielmehr, wie schon bei den Messungen im Ethernet erkannt wurde, von der Verarbeitungzeit des ServiceRegistry dominiert. Die Verarbeitungszeit wird wiederum hauptsächlich von der ServiceRegistry-Größe bestimmt.

Als Fazit kann man sagen, dass die Anmeldung eines Anwendungsdienstes am ServiceRegistry auch in Wireless LAN-basierten Netzwerkumgebungen sehr performant abgewickelt werden kann und praktisch den sehr guten Verhältnissen beim Ethernet gleichwertig ist.

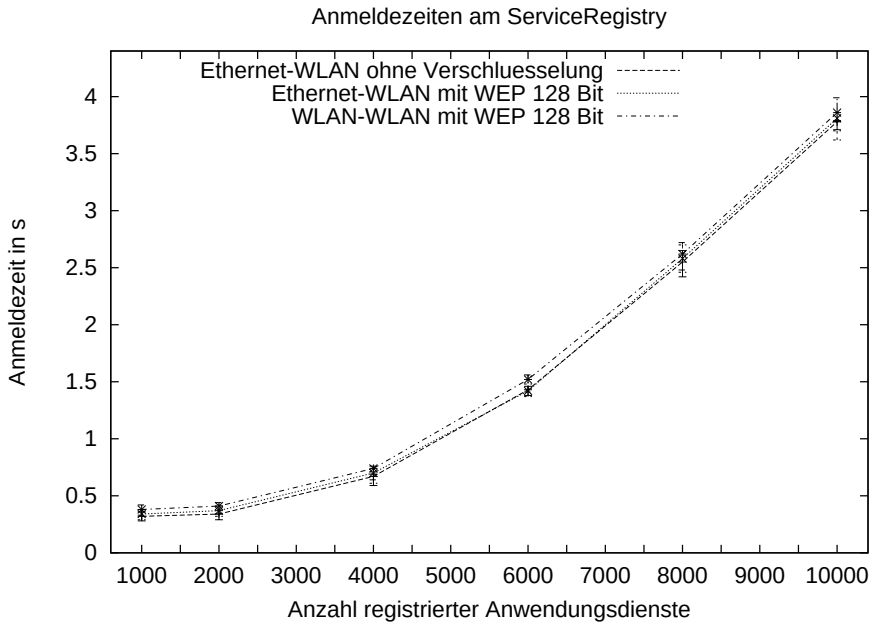

Abbildung 12.7: Die Anmeldezeit eines Anwendungsdienstes am ServiceRegistry in verschiedenen, WLAN-basierten Netzwerkumgebungen in Abhängigkeit von der Anzahl seiner registrierten Dienste

12.4 Zusammenfassung

Die Ergebnisse der Evaluation ServiceJugglers zeigen, dass ein lokales logisches Netzwerk wie QuickLink auf der Ebene der Anwendungsdienste performant durch einen Peer, der eine herausgehobene Rolle annimmt und zentrale Managementaufgaben übernimmt, verwaltet werden kann. Dabei leistet der herausgestellte Peer als Regions-Manager zusätzliche Dienste, von denen der ServiceRegistry als Verzeichnisdienst für Anwendungsdienste der wichtigste ist. In der Evaluation wurde nachgewiesen, dass der ServiceRegistry und mit ihm die Rolle des Regions-Managers in dynamischen Situationen auf potentere QuickLink-Knoten migrierbar ist. Die Migration selbst ist unter realistischen Annahmen und Bedingungen performant umsetzbar. Ebenso ist die Anmeldung von Anwendungsdiensten am ServiceRegistry in performanter Weise möglich, so dass die durch QuickLink geleistete Abbildung der Netzwerkdynamik effizient unterstützt wird.

13 APLICOOVER

Die bisher evaluierten Komponenten QuickLinkNets *QuickLink* und *ServiceJuggler* realisieren die lokale Sicht auf das Netzwerk. Daher decken ihre Instanzen nur jeweils einen lokal begrenzten Bereich des gesamten logischen Netzwerkes ab, nämlich eine Region. APLICOOVERs Aufgabe in QuickLinkNet liegt dagegen in der Vernetzung der Regionen zu einem Ganzen. Dieses Ganze ist ein logisches Netzwerk mit dem Potential zu einer globalen Ausdehnung. APLICOOVER realisiert somit die globale Sicht auf das Netzwerk, welche einen globalen Informationsraum aus Anwendungsdiensten und ihren Zugriffspunkten im Internet darstellt.

APLICOOVER bildet alle notwendigen Funktionen ab, um ein solches globales logisches Netzwerk zu verwirklichen. Es bedient sich dabei des Peer-to-Peer-Systems P-Grid [Abe01], dessen Technologie und Funktionalität es nutzt, um eigene, auf QuickLinkNet zugeschnittene Funktionen und Dienste zu realisieren. APLICOOVER kapselt P-Grid komplett ein, so dass andere Komponenten auf dessen Funktionen keinen direkten Zugriff haben.

Die Systemeigenschaften APLICOOVERs werden zum größten Teil vom Verhalten der unterliegenden P-Grid-Schicht bestimmt. Zum Test des Verhaltens eines großen, P-Grid-basierten Netzwerkes mit globaler Ausdehnung ist jedoch ein Testbed mit wenigstens mehreren hundert, weltweit verteilten P-Grid-Knoten notwendig. Da APLICOOVER die Verwaltungseinheit in der P-Grid-Schicht von einem einzelnen Knoten auf eine ganze Region ausdehnt, wäre für ein entsprechendes QuickLinkNet sogar ein Testbed mit mehreren hundert Regionen notwendig. Auf einen solchen Test wurde verzichtet und auf einen späteren Zeitpunkt verschoben, da er den Rahmen dieser Dissertation gesprengt hätte. APLICOOVER wurde jedoch umfassenden Tests unterzogen, um die einwandfreie Funktionalität der Software und ihr Zusammenspiel mit P-Grid sicherzustellen. Diese werden in den nächsten Abschnitten beschrieben.

13.1 Testumgebung

Die Tests APLICOOVERs fanden mit vier Testrechnern in einem Ethernet nach IEEE 802.3u mit einer Datenrate von 100 MBit/s im Betriebsmodus Vollduplex statt. Als Messrechner dienten die aus den Messungen der anderen Komponenten QuickLinkNets bekannten Rechner *Toshiba*, *Gericom*, *Thinkpad* und

Silentbob, deren technische Daten in Tabelle 11.1 auf Seite 201 angegeben sind. Als Netzwerkgerät wurde der in der Tabelle 11.2 aufgeführte Switch/Router *SMC 7004ABR* verwendet.

Da das Testnetzwerk die globalen Verbindungen zwischen Regionen simulieren sollte, wurde auf einen Start von QuickLink auf den Messrechnern verzichtet. Vielmehr wurde jeweils ein QuickLink-Dummy als Datenbasis für ServiceJuggler verwendet, welches diesen jeweils ein eigenes, stabiles QuickLink-Netzwerk simulierte. Auf jedem Messrechner wurde eine modifizierte ServiceJuggler-Instanz in der Rolle des Regions-Managers gestartet, welche während des Tests mit seiner APLICOOVER-Instanz kommunizieren sollte. Da mehrere ServiceJuggler in einem technischen Netzwerk laufen sollten, waren diese so modifiziert, dass sie nicht miteinander kommunizieren konnten und so annehmen mussten, dass sie als ServiceJuggler-Instanz jeweils allein im Netzwerk waren. Der ServiceRegistry eines jeden ServiceJugglers wurde mit generierten Servicedaten vorkonfiguriert, um ein stabiles Regionsprofil liefern zu können. Die Ausführung des Tests wurde auf jedem Messrechner durch ein Testskript realisiert, welches eine verteilte Anwendung, wie ein MA es darstellt, simulierte und die entsprechenden Methoden APLICOOVERs ansteuerte.

13.2 Testmethodik

Der Ausgangszustand des Tests war auf jedem Messrechner eine laufende, modifizierte ServiceJuggler-Instanz mit QuickLink-Dummy, welche als Regions-Manager fungierte. Das im vorherigen Abschnitt beschriebene Testskript initiierte nun nacheinander die folgenden Vorgänge:

1. Starten einer APLICOOVER-Instanz und Registrieren in der Java RMI Registry.
2. Initialisierung der APLICOOVER-Instanz.
3. Abrufen und Veröffentlichen des Regionsprofils.
4. Starten einer Dienstabfrage, die alle veröffentlichten Dienstinstanzen zurückliefert.
5. Starten einer Dienstabfrage, die genau eine zutreffende veröffentlichte Dienstinstanz zurückliefert.
6. Modifizieren eines veröffentlichten Diensteintrages.
7. Löschen eines veröffentlichten Diensteintrages.
8. Starten einer Dienstabfrage nach dem modifizierten Diensteintrag.

9. Starten einer Dienstabfrage nach dem gelöschtem Diensteintrag.
10. Starten einer Bereichssuche.
11. Starten einer gecachten Dienstabfrage mit *cache-hit*.
12. Starten einer gecachten Bereichssuche mit *cache-miss*, die im "globalen" APLICOOVER-Netzwerk fortgesetzt wird.
13. Herunterfahren von APLICOOVER.

Nach den ersten beiden Schritten legte das Testskript eine Pause von 3 Minuten ein, um den P-Grid-Instanzen in APLICOOVER die Möglichkeit zu geben, in die Replikationsphase zu treten. In dieser Phase tauschten die P-Grid-Instanzen Nachrichten aus, um ihre Position in der binärbaumbasierten Struktur P-Grids zu finden und ihre Routingtabellen entsprechend anzupassen. Nach der Wartezeit von 3 Minuten, die zuvor experimentiell ermittelt worden war, konnte sichergestellt werden, dass sich das P-Grid-Netzwerk in einem stabilen Zustand befand und alle APLICOOVER-Instanzen vollständig arbeitsfähig waren.

Nun riefen, ausgelöst durch Schritt 3 des Testskriptes, die APLICOOVER-Instanzen das Regionsprofil ihres ServiceJugglers ab und veröffentlichten dieses im "globalen" Netzwerk. Die P-Grid-Instanzen gerieten nun in ihre Bootstapping-Phase, in der die Diensteinträge der veröffentlichten Regionsprofile entsprechend der entstandenen Netzwerkstruktur auf die vorhandenen Knoten verteilt wurden. Dazu wurde den Knoten wiederum 3 Minuten Zeit gelassen. Es folgten nun die restlichen Schritte im Testskript, welche nacheinander ausgeführt wurden.

13.3 Ergebnisse

Jeder Testschritt und jede Funktion APLICOOVERs endete mit den zu erwarteten Ergebnissen. Die Methodenaufrufe des Testskripts können im Anhang A.4 nachgelesen werden, ebenso die Ausgaben des Testbeds als Reaktion auf die Aufrufe. Wie sich aus den Ausgaben entnehmen lässt, funktionieren vor allem die Suchmethoden APLICOOVERs so wie erwartet. Die gefundenen Ergebnisse entsprachen genau den Vorgaben, welche die Testdaten erwarten ließen. Es gab beim Test keinerlei Probleme, so dass die Erwartungen an die Funktionalität APLICOOVERs voll erfüllt wurden.

13.4 Zusammenfassung

APLICOOVERs Tests verliefen wie erwartet und wiesen nach, dass die Software im praktischen Betrieb anwendbar ist und ihre Aufgaben erfüllt. Die wichtigste Komponente in APLICOOVER stellt die P-Grid-Schicht dar, über welches

die Vernetzung im globalen Internet realisiert wird. P-Grid [Abe01] kann auf eine lange Entwicklungszeit zurückblicken und ist entsprechend ausgereift und gut dokumentiert. Ständige Weiterentwicklungen [DHA03, HDA03, ADH04b, ADH04a, ADH05] P-Grids und dessen Anwendung als unterliegende Technologie in anderen Projekten [Sch02, ACMDH03, HS05] untermauern seine Bedeutung und Anwendbarkeit. Dazu gehören auch Tests und Messungen in größeren Testbeds mit bis zu 400 weltweit verteilten Knoten [ADHS05, KSHS07], welche P-Grids Leistungsfähigkeit, Skalierbarkeit und Zuverlässigkeit im globalen Internet nachweisen.

Unter diesen Bedingungen und im Ergebnis des erfolgreichen Tests der Softwarekomponente APLICOOVER kann davon ausgegangen werden, dass APLICOOVER seine Aufgabe der globalen Vernetzung der Regionen QuickLinkNets unter Einhaltung der Mengenskalierbarkeit der zu registrierenden Anwendungsdienste auch unter den Bedingungen des globalen Internets erfolgreich erfüllt.

14 Evaluation der Thesen

Die Evaluation der Komponenten QuickLinkNets hat gezeigt, dass die praktische Umsetzung der in dieser Dissertation entwickelten Architektur und der aufgestellten Thesen möglich ist.

Struktur Die Analyse der Defizite der Infrastrukturen bestehender logischer Netzwerke bezüglich der Anforderungen von MAS der Stufe 2 im Kapitel 4 und der daraus entwickelte Lösungsweg in Kapitel 5 dieser Dissertation mündeten in einem Architekturvorschlag in Abschnitt 5.3. Diese Architektur sieht drei Komponenten vor, mit welchen die verschiedenen, sich zum Teil diametral gegenüberliegenden Zieleigenschaften des gesamten logischen Netzwerkes erfüllt werden können. In der Umsetzung der Architekur entsteht so ein globales logisches Netzwerk, welches nach zwei logischen Sichten, der lokalen und der globalen Sicht, strukturiert ist. QuickLinkNet stellt in seiner prototypischen Umsetzung die Referenzimplementierung des Architekturvorschlags dar. Dabei wird auf jeder Schicht eine eigene Organisationsstruktur verwendet, mit der QuickLinkNet die jeweiligen Zieleigenschaften umsetzen kann. Die lokale Sicht mit ihrer Adaptionsfähigkeit an hochdynamische Netzwerkvorgänge wird durch die Komponenten QuickLink und ServiceJuggler in Form der Region realisiert, während die globale Sicht, d.h. die Verbindung aller Regionen, durch APLICOOVER erfüllt wird. Die intelligente Verbindung der beiden Sichten wird ebenfalls durch die Komponente ServiceJuggler geleistet. Damit ist die grundlegende These 1 bestätigt:

These 1 - Struktur: Die Trennung des gesamten mobilen Agentensystems in eine globale und eine lokale Sicht hilft, alle Anforderungen an eine Infrastruktur für MAS der Stufe 2 zu erfüllen. Für die Verbindung der beiden Sichten ist eine intelligente Komponente nötig, welche die autonome Verwaltung des lokalen Bereiches sicherstellt und eine Selbstadaption des Netzwerkes an dynamische Vorgänge ermöglicht.

Dynamik Mit der Evaluation der Komponente QuickLink in Kapitel 11 konnte gezeigt werden, dass der Eintritt hochdynamischer Plattformen in einem lokalen logischen Netzwerk im Sekundenbereich geleistet werden kann und auch bei hoher Netzwerkdynamik keine hohen Netzwerkbelastungen entstehen müssen. Es wurde ebenfalls nachgewiesen, dass die Erweiterung der Betrachtungsweise von

Peers bezüglich konkreter Leistungsparameter möglich ist. Diese Leistungsparameter werden von der Komponente ServiceJuggler zur Steuerung der Selbstadaption einer Region herangezogen. Dass diese Adaption ebenfalls performant ausführbar ist, konnte in der Evaluation ServiceJugglers in Kapitel 12 gezeigt werden. Somit ist die These 2 erfüllt:

These 2 - Dynamik: Mit einer lokalen Sicht auf ein mobiles Agentensystem der Stufe 2 ist es möglich, auch hohe Netzwerkdynamik performant abzubilden. Darüber hinaus brauchen mobile Agenten der Stufe 2 eine intelligente Unterstützung seitens ihrer umgebenden Infrastruktur, die über die gewöhnliche Transparenz im Peer-to-Peer-System hinaus geht.

Skalierbarkeit APLICOOVER realisiert die globale Sicht auf ein mobiles Agentensystem, indem es die Regionen auf der Basis ihrer registrierten Anwendungsdienste und deren Zugangspunkte mit Hilfe einer DHT-basierten Struktur miteinander verbindet. Die Evaluation der Komponente APLICOOVER ergab deren Funktionsfähigkeit und, beruhend auf anderen Arbeiten, die Skalierbarkeit, Performance und Zuverlässigkeit ihrer wichtigsten funktionalen Schicht P-Grid in globalen logischen Netzwerken. Folglich ist auch die These 3 bestätigt:

These 3 - Skalierbarkeit: Die globale Sicht auf ein mobiles Agentensystem hilft, die globale Skalierbarkeit des Gesamtsystems zu beherrschen. Diese kann durch einen DHT-basierten Ansatz realisiert werden.

Praktikabilität QuickLinkNets prototypische Implementierung erfolgte modular, so dass die Komponenten des Frameworks wie auch einige funktionale Bausteine der Komponenten einzeln nach Bedarf gestartet werden können. Die Implementierung erfolgte in Java 2 Standard Edition 5.0 [Sun07b], so dass sie auf jeder mit der entsprechenden Laufzeitumgebung ausgerüsteten Plattform lauffähig sind. QuickLinkNet bietet seine Funktionen über Schnittstellen an, so dass sie in jede kompatible Anwendung eingebunden werden kann. Ferner konnte für die Komponente APLICOOVER eine schon existierende Technologie, P-Grid, verwendet und eingebunden werden. Damit konnte gezeigt werden, dass die Praktikabilität der Umsetzung und damit die These 4 erfüllt ist:

These 4 - Praktikabilität: Die Implementierung der genannten Komponenten ist möglich, ohne Schnittstellen zu vorhandenen Systemen durchbrechen zu müssen. Darüber hinaus ist es für eine der Komponenten sogar möglich, schon vorhandene Technologien zu nutzen.

15 Zusammenfassung und Ausblick

In dieser Dissertation wurde das Infrastrukturdienst-Framework QuickLinkNet vorgestellt. Es dient als dienstorientierte Infrastruktur für verteilte Anwendungen, wie sie mobile Agentensysteme darstellen. Bei bisherigen Lösungen konnten die diametral entgegengesetzt wirkenden Anforderungen an eine solche Infrastruktur, nämlich Aktualität der Dienstinformationen auch in hochdynamischen Umgebungen und gleichzeitig globale Skalierbarkeit bezüglich der abbildbaren Dienstanzahl und der Nachrichtenzahl, nicht erfüllt werden. Diese Anforderungen sind aber gerade bei mobilen Agentensystemen der Stufe 2, die im Mittelpunkt unserer Betrachtungen stehen, ein MUSS.

Ausgangspunkt der Arbeit war das Problem, dass bestehende Systeme meist versuchen, die Transparenz der Internetschicht und den Gedanken des reinen Peer-to-Peer-Systems aufrecht zu erhalten. Damit sind sie gezwungen, das logische Netzwerk als Basis der dienstorientierten Infrastruktur mit *einer* universellen Strukturierungsmethode aufzubauen, welche auch die Systemeigenschaften weitgehend festlegt. Existierende Lösungen für verteilte, dienstorientierte Infrastrukturen betonen daher *entweder* die globale Skalierbarkeit *oder* die Aktualität der Informationen. Dieser Konflikt sollte gelöst werden.

Die hauptsächliche These dieser Dissertation fordert daher die organisatorische Trennung eines verteilten Systems in zwei Sichten, eine lokale und eine globale Sicht. Jede Sicht bildet einen Teil des gesamten logischen Netzwerkes ab, welcher aber mit jeweils einer eigenen Organisationsstruktur aufgebaut wird. Damit ist in jeder Sicht eine jeweils andere Betonung der Systemeigenschaften und in ihrer Gesamtheit die Erfüllung der gewünschten gegensätzlichen Systemeigenschaften möglich. Im ersten Teil dieser Dissertation wurden dazu die Eigenschaften, Abhängigkeiten und Optionen bestehender verteilter Systeme und die Anforderungen mobiler Agentensysteme der Stufe 2 an eine dienstorientierte Infrastruktur herausgearbeitet. Die Analyse der untersuchten Systeme ergab dabei, dass keines den Anforderungen an die Zieleigenschaften einer solchen Infrastruktur gerecht wird und eine globale Skalierbarkeit leistet, ohne auf die Abbildbarkeit hochdynamischer Agentenplattformen verzichten zu müssen.

Der zweite Teil der Dissertation umfasst alle Arbeiten, welche die Lösung der in Teil 1 aufgezeigten Problematik behandeln. Anhand der aufgezeigten Probleme und der gewünschten Zieleigenschaften einer Infrastruktur für mobile Agentensysteme der Stufe 2 wurden abgeschlossene Bereiche von Systemeigenschaften aufgestellt und so strukturiert, dass sie sich einzeln lösen lassen. Aus

diesen einzelnen Lösungsbereichen wurde jeweils eine Softwarekomponente mit den zugehörigen Zieleigenschaften formuliert und diese in einer Referenzarchitektur aus Infrastrukturdiensten zusammengeführt. Die Beschreibung der prototypischen Implementierung der Referenzarchitektur als Infrastrukturdienstframework *QuickLinkNet* mit seinen drei Komponenten *QuickLink*, *ServiceJuggler* und *APLICOOVER* vervollständigt den Teil 2. Hier wurde im Bereich der lokalen Sicht, die durch QuickLink und ServiceJuggler gebildet wird, eine abbildbare Dynamik im Sekundenbereich erreicht. Auf Grundlage der Erweiterung der Transparenz von Peers bezüglich ihrer Leistungseigenschaften konnte ferner ein proaktives Selbstadaptionsverhalten innerhalb eines Region genannten lokalen Bereichs des logischen Netzwerkes implementiert werden. Im globalen Bereich wurde ein selbstorganisierendes, DHT-basiertes System etabliert, mit dem eine globale Skalierbarkeit sichergestellt werden kann.

Mit der praktischen Evaluation QuickLinkNets in Teil 3 der Dissertation konnte der Nachweis erbracht werden, dass alle gewünschten Systemeigenschaften einer Infrastruktur für ein mobiles Agentensystem der Stufe 2, wie abbildbare Dynamik, Aktualität der Informationen und globale Skalierbarkeit bezüglich der Menge der abbildbaren Dienste der dafür nötigen Nachrichten, in *einem* Infrastrukturframework erreichbar sind. Damit wurde gezeigt, dass mobile Agentensysteme seitens ihrer dienstorientierten Infrastruktur so performant unterstützt werden können, dass ein autonomes Routing mobiler Agenten und damit allgemein ein geplantes Vorgehen intelligenter Softwareentitäten in einer dienstorientierten Umgebung ebenfalls performant umsetzbar wird.

Zukünftige Arbeiten werden sich auf die Verbesserung der bisher prototypischen Implementierung der Komponenten QuickLinkNets richten. In QuickLink soll vor allem die Überarbeitung und das Feintuning der inneren Abläufe in Zukunft zusätzliche Laufstabilität bringen. Ferner soll, aus den Erfahrungen der Messungen in funkgestützten Netzwerken resultierend, die Robustheit des QuickLink-Netzwerkes weiter verbessert werden. Eine Weiterentwicklung der Messmethoden zur Leistungsmessung soll eine bessere Vergleichbarkeit der Messergebnisse in Linux- und Windows-basierten Systemen sicherstellen. Die Komponente ServiceJuggler soll in Zukunft befähigt werden, zur Regionssteuerung mehrere Leistungsparameter kombiniert auszuwerten. Für eine flexiblere Regionssteuerung sollen seine Auswertealgorithmen in einer Skriptsprache implementiert werden, um die Einbindung neuer Algorithmen ohne Abhängigkeit vom Java-Quellcode zu ermöglichen. Beim Verzeichnisdienst ServiceRegistry sind ebenfalls Verbesserungen möglich, um die ungünstige, nichtlineare Abhängigkeit der Datengröße von der Anzahl der registrierten Dienste zu mildern. Ganz klar werden sich unsere nächsten Aktivitäten auch auf den Test QuickLinkNets in einem großen Testbed mit globaler Ausdehnung richten, um mehr Erfahrungen mit dem Infrastrukturdienst-Framework zu sammeln.

Literaturverzeichnis

[A+07] Aberer, Karl u. a.: *The P-Grid Project - Publications - Applications.* http://www.p-grid.org/publications/applications.html. Version: 2007

[Abe01] Aberer, Karl: P-Grid: A Self-Organizing Access Structure for P2P Information Systems. In: *Sixth International Conference on Cooperative Information Systems (CoopIS 2001).* Trento, Italy, September 5–7 2001

[Abe07] Aberer, Karl: *P2P Systems Research of Karl Aberer.* http://lsirpeople.epfl.ch/aberer/P2P.htm. Version: 2007

[ACMDH03] Aberer, Karl ; Cudré-Mauroux, Philippe ; Datta, Anwitaman ; Hauswirth, Manfred: PIX-Grid: A Platform for P2P Photo Exchange. In: *Proceedings of Ubiquitous Mobile Information and Collaboration Systems (UMICS 2003).* Klagenfurt/Velden, Austria, 16-17 June 2003

[ADH04a] Aberer, Karl ; Datta, Anwitaman ; Hauswirth, Manfred: Efficient, self-contained handling of identity in Peer-to-Peer systems. In: *IEEE Transactions on Knowledge and Data Engineering* 16 (2004), July, Nr. 7

[ADH04b] Aberer, Karl ; Datta, Anwitaman ; Hauswirth, Manfred: Route maintenance overheads in DHT overlays. In: *6th Workshop on Distributed Data and Structures (WDAS'2004).* Lausanne, Switzerland, July 8-9 2004

[ADH05] Aberer, Karl ; Datta, Anwitaman ; Hauswirth, Manfred: A decentralized public key infrastructure for customer-to-customer ecommerce. In: *International Journal of Business Process Integration and Management* Volume 1 (2005), Nr. 1

[ADHS05] Aberer, K. ; Datta, A. ; Hauswirth, M. ; Schmidt, R.: Das P-Grid-Overlay-Netzwerk: Von einem einfachen Prinzip zu einem komplexen System. In: *Datenbank Spektrum* 5(13) (2005), S. 5–14

[Agl04] *Aglets.* http://aglets.sourceforge.net. Version: 2004

[AH02] Aberer, Karl ; Hauswirth, Manfred: *An Overview of Peer-to-Peer Information Systems.* citeseer.ist.psu.edu/aberer02overview.html. Version: 2002

[AHJN04] Antoniu, Gabriel ; Hatcher, Phil ; Jan, Mathieu ; Noblet, David A.: *Performance Evaluation of JXTA Communication Layers.* http://www.inria.fr/rrrt/rr-5469.html. Version: Oktober 2004

[ALk05] *AgentLink – European Co-ordination Action for Agent Based Computing.* http://www.agentlink.org/. Version: 2005

[ANS07] ANSI/IEEE Std 802.11 1999 Edition: *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.* http://grouper.ieee.org/groups/802/11/index.html. Version: 2007

[APHS02] Aberer, K. ; Punceva, M. ; Hauswirth, M. ; Schmidt, R.: Improving Data Access in P2P Systems. In: *IEEE Internet Computing* (2002), Nr. 6(1), S. 58–67

[ASW+99] Arnold, Ken ; Scheifler, Robert ; Waldo, Jim ; O'Sullivan, Bryan ; Wollrath, Ann ; O'Sullivan, B. ; Wollrath, A.: *Jini Specification*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1999. – ISBN 0201616343

[B+03] Braun, Peter u. a.: *Tracy – The Mobile Agent System*. http://tracy.informatik.uni-jena.de. Version: 2003

[BCCS99] Bellavista, P. ; Cavallari, C. ; Corradi, A. ; Stefanelli, C.: Mobile Agents for Internet Services: Directions of Standardization and their Implementation in SOMA. In: *Proceedings of the 37th Conference of the Associazione Italiana per l'Informatica ed il Calcolo Automatico (AICA'99)*, 1999, S. 19–31

[BCMT03] Bisignano, Mario ; Calvagna, Andrea ; Modica, Giuseppe D. ; Tomarchio, Orazio: Expeerience: A Jxta Middleware for Mobile Ad-Hoc Networks. In: *P2P '03: Proceedings of the 3rd International Conference on Peer-to-Peer Computing*. Washington, DC, USA : IEEE Computer Society, 2003. – ISBN 0–7695–2023–5, S. 214

[BCPR03] Bellifemine, F. ; Caire, G. ; Poggi, A. ; Rimassa, G.: *JADE Whitepaper*. http://jade.tilab.com/papers/WhitePaperJADEEXP.pdf. Version: 2003

[BCS01] Bellavista, Paolo ; Corradi, Antonio ; Stefanelli, Cesare: Mobile Agent Middleware for Mobile Computing. In: *IEEE Computer* 34 (2001), Nr. 2, S. 73–81

[Bea05] *BearShare*. http://bearshare.com. Version: 2005

[BER00] Braun, Peter ; Erfurth, Christian ; Rossak, Wilhelm: An Introduction to the Tracy Mobile Agent System / Friedrich-Schiller-Universität Jena, Institut für Informatik. 2000 (Math/Inf/00/24). – Forschungsbericht

[BER01] Braun, Peter ; Eismann, Jan ; Rossak, Wilhelm: Managing Tracy Agent Server Networks / Friedrich-Schiller-Universität Jena, Institut für Informatik. 2001 (Technical Report 01/12). – Forschungsbericht

[BL04] Buccafurri, Francesco ; Lax, Gianluca: TLS: A Tree-Based DHT Lookup Service for Highly Dynamic Networks. In: *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE, Proceedings, Part I, Lecture Notes in Computer Science LNCS 3290*. Agia Napa (Zypern) : Springer Verlag, Berlin Heidelberg, Oktober 2004, S. 563–580

[Bol05] Bologna Universita di: *SOMA: Secure and Open Mobile Agent*. http://lia.deis.unibo.it/Research/SOMA. Version: 2005

[BR05] Braun, Peter ; Rossak, Wilhelm R.: *Mobile Agents - Basic Concepts, Mobility Models, & The Tracy Toolkit*. Heidelberg : dpunkt.verlag, 2005

[Bra89] Braden, R.: *Requirements for Internet Hosts – Communication Layers*. http://www.ietf.org/rfc/rfc1122.txt?number=1122. Version: Oktober 1989. – RFC 1122

[Bra03] Braun, Peter: *The Migration Process of Mobile Agents - Implementation, Classification, and Optimization*, Friedrich-Schiller-Universität Jena, Diss., 2003

[CDK02] Coulouris, George ; Dollimore, Jean ; Kindberg, Tim: *Verteilte Systeme - Konzepte und Design*. 3., überarbeitete Auflage. München : Pearson Education Deutschland GmbH, 2002

[CFZ03] Choi, J. H. ; Franke, H. ; Zeilenga, K.: Enhancing the Performance of OpenLDAP Directory Server with Multiple Caching. In: *Proceedings of International Symposium on Performance Evaluation of Computers and Telecommunication Systems (SPECTS)*, 2003. – ISBN 1–56555–269–5, S. 737–744

[Chi] Chiusoli, Claudia: *Dynamic Configuration, Master Thesis*. http://www-lia.deis.unibo.it/Research/SOMA/chiusoli.pdf. Version: ?

[CSWH01] Clarke, I. ; Sandberg, O. ; Wiley, B. ; Hong, T.: Freenet – A Distributed Anonymous Information Storage and Retrieval System. In: *Proceedings of Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability* Bd. LLNCS 2009, Springer Verlag, 2001

[DA99] Dierks, T. ; Allen, C.: *The TLS Protocol Version 1.0*. http://www.ietf.org/rfc/rfc2246.txt?number=2246. Version: January 1999

[Dat07] Datta, Anwitaman: *Publications of Anwitaman Datta*. http://www.ntu.edu.sg/home/anwitaman/Publications.html. Version: 2007

[DBK+01] Dabek, Frank ; Brunskill, Emma ; Kaashoek, M. F. ; Karger, David ; Morris, Robert ; Stoica, Ion ; Balakrishnan, Hari: Building Peer-to-Peer Systems with Chord, a Distributed Lookup Service. In: *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, 2001

[DER05] Döhler, Arndt ; Erfurth, Christian ; Rossak, Wilhelm: A Framework of Autonomous and Self-adaptable Middleware Services to Support Mobile Agents in Dynamic Networks. In: *GESTS International Transactions on Computer Science and Engineering* Volume 19 (2005), Oktober, Nr. No. 1, S. 109 – 122

[DH98] Deering, S. ; Hinden, R.: *Internet Protocol Version 6 (IPv6)*. http://www.faqs.org/rfcs/rfc2460.html. Version: Dezember 1998. – RFC 2460

[DHA03] Datta, Anwitaman ; Hauswirth, Manfred ; Aberer, Karl: Updates in Highly Unreliable, Replicated Peer-to-Peer Systems. In: *The 23rd International Conference on Distributed Computing Systems, USA*, 2003

[DHW02] D. Harrington, R. P. ; Wijnen, B.: *An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks*. http://tools.ietf.org/html/rfc3411. Version: 2002. – RFC 3411

[Dro97] Droms, R.: *Dynamic Host Configuration Protocol*. http://www.faqs.org/rfcs/rfc2131. Version: 1997. – RFC 2131

[EDR04] Erfurth, Christian ; Döhler, Arndt ; Rossak, Wilhelm R.: A First Look at the Performance of Autonomous Mobile Agents in Dynamic Networks. In: *Proceedings of the Thirty-Seventh Hawaii International Conference on System Sciences (HICSS-37)*. Big Island, Hawaii, USA, 5-8 January, 2004

[EPC07a] EPCC: *Unparalleled Computing*. http://www.epcc.ed.ac.uk/. Version: 2007

[EPC07b] EPCC, The University of Edinburgh: *The Java Grande Forum Benchmark Suite*. http://www2.epcc.ed.ac.uk/computing/research_activities/java_grande/index_1.html. Version: 2007

[ER02] Erfurth, Christian ; Rossak, Wilhelm: Characterization and Management of Dynamical Behaviour in a System With Mobile Agents. In: Unger, Herwig (Hrsg.) ; Böhme, Thomas (Hrsg.) ; Mikler, Armin (Hrsg.): *Innovative Internet Computing System - Second International Workshop, IICS 2002, Kühlungsborn (Germany), June 2002* Bd. 2346, Springer, 2002 (lncs), S. 109–119

[Erf04] Erfurth, Christian: *Proaktive autonome Navigation für mobile Agenten - Ein Schritt in Richtung mobiler Agentensysteme der nächsten Generation*, Friedrich-Schiller-Universität Jena, Diss., 2004

[FFMM94] Finin, T. ; Fritzson, R. ; McKay, D. ; McEntire, R.: KQML as an Agent Communication Language. In: Adam, N. (Hrsg.) ; Bhargava, B. (Hrsg.) ; Yesha, Y. (Hrsg.): *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*. Gaithersburg, MD, USA : ACM Press, 1994, 456–463

[FKK96] Freier, Alan O. ; Karlton, Philip ; Kocher, Paul C.: *The SSL Protocol Version 3.0*. http://wp.netscape.com/eng/ssl3/draft302.txt. Version: November 1996

[Fou99] Foundation for Intelligent Physical Agents (FIPA): *OMG-FIPA Liaison - DRAFT*. http://www.objs.com/isig/omg-fipa-liaison-4.html. Version: 1999

[Fou01a] Foundation for Intelligent Physical Agents: *FIPA Agent Software Integration Specification*. http://www.fipa.org/specs/fipa00079/XC00079B.html. Version: 2001

[Fou01b] Foundation for Intelligent Physical Agents: *FIPA Policies and Domains Specification*. http://www.fipa.org/specs/fipa00089/PC00089D.html. Version: 2001

[Fou02a] Foundation for Intelligent Physical Agents: *FIPA Abstract Architecture*. http://www.fipa.org/specs/fipa00001/SC00001L.html. Version: 2002

[Fou02b] Foundation for Intelligent Physical Agents: *FIPA ACLMessage Representation in XML Specification*. http://www.fipa.org/specs/fipa00071/SC00071E.html. Version: 2002

[Fou02c] Foundation for Intelligent Physical Agents: *FIPA Quality of Service Ontology Specification*. http://www.fipa.org/specs/fipa00094/SC00094A.html. Version: 2002

[Fou07] Foundation for Intelligent Physical Agents: *FIPA Specifications*. http://www.fipa.org/repository/standardspecs.html. Version: 2007

[Fra07] Frangella, Emilio: *CPUBench2003, Version 1.5 Beta 2*. http://www.hwupgrade.it/download/file/1683.html. Version: 2007

[FZ01] Fieger, Andreas ; Zitterbart, Martina: Zuverlässige Transportdienste für Mobile Computing. In: *Inform., Forsch. Entwickl.* 16 (2001), Nr. 4, S. 179–188

[Gha00] Ghaffar, Atif: *LDAP unter Linux - Einführung*. http://www.linuxfocus.org/Deutsch/July2000/article159.shtml. Version: 2000

[GLD+05] Grow, Robert M. ; Law, David J. ; Diab, Wael W. ; Carlson, Steven B. ; Dawe, Piers: *Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*. http://standards.ieee.org/getieee802/download/802.3-2005_section1.pdf. Version: 2005

[Gon02] Gong, Li: *Project JXTA: A Technology Overview*. http://www.jxta.org/project/www/docs/TechOverview.pdf. Version: Oktober 2002

[Gro05] *Grokster*. http://www.grokster.com. Version: 2005

[Hau07] Hauswirth, Manfred: *Publications of Manfred Hauswirth*. http://www.manfredhauswirth.org/publications/recent-work/recent-work.html. Version: 2007

[HD05] Hauswirth, M. ; Dustdar, S.: Peer-to-Peer: Grundlagen und Architektur. In: *Datenbank-Spektrum* (2005), May, Nr. 13, S. 5–13. – dpunkt.verlag

[HDA03] Hauswirth, Manfred ; Datta, Anwitaman ; Aberer, Karl: Handling Identity in Peer-to-Peer Systems. In: *Proceedings of the 6th International Workshop on Mobility in Databases and Distributed Systems, in conjunction with the 14th International Conference on Database and Expert Systems Applications (DEXA'2003)*. Prague, Czech Republic, September 1-5 2003

[Hen06] Hentrich, Thomas: *A realization of the global component for the MAS-supporting QuickLink middleware*, Friedrich-Schiller-Universität Jena, Fakultät für Mathematik und Informatik, Institut für Informatik, Lehrstuhl für Softwaretechnik, Diplomarbeit, June 2006

[HPS00] Häckelmann, H. ; Petzold, H.J. ; Strahinger, S.: *Kommunikationssysteme — Technik und Anwendungen*. Springer-Verlag, 2000

[HS05] Hauswirth, Manfred ; Schmidt, Roman: An overlay network for resource discovery in Grids. In: *Second International Workshop on Grid and Peer-to-Peer Computing Impacts on Large Scale Heterogeneous Distributed Database Systems in conjunction with the 16th International Conference on Database and Expert Systems Applications*. Copenhagen, Denmark, 22 - 26 August 2005

[Hub07] Hubert, Bert: *Linux Advanced Routing & Traffic Control*. http://www.lartc.org/. Version: 2007

[Int07] Internet Systems Consortium: *Host count history*. http://www.isc.org/index.pl?/ops/ds/host-count-history.php. Version: Januar 2007

[ISO05] *International Organization for Standardization*. http://www.iso.org/. Version: 2005

[JAN02] JANNOTTI, J.: Network layer support for overlay networks. In: *IEEE OPENARCH*, 2002, S. 3–13

[Jav07a] Java RMI: *Java™ Remote Method Invocation (Java RMI)*. http://java.sun.com/j2se/1.5.0/docs/guide/rmi/. Version: 2007

[Jav07b] JAVA VM: *Java^{TM} Virtual Machines*. http://java.sun.com/j2se/1.5.0/docs/guide/vm/index.html. Version: 2007

[JPA04] JOHNSON, D. ; PERKINS, C. ; ARKKO, J.: *Mobility Support in IPv6*. http://www.faqs.org/rfcs/rfc3775.html. Version: 2004. – RFC 3775

[JW98] JENNINGS, Nicholas R. (Hrsg.) ; WOOLDRIDGE, Michael J. (Hrsg.): *Agent technology: foundations, applications, and markets*. Secaucus, NJ, USA : Springer-Verlag New York, Inc., 1998. – ISBN 3–540–63591–2

[Jxt01] *P2P-Protokoll JXTA als Open Source freigegeben*. http://www.heise.de/newsticker/meldung/17345. Version: April 2001

[Kan01] KAN, Gene: Gnutella. In: ORAM, Andy (Hrsg.): *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*. first. O'Reilly and Associates, March 2001, S. 94–122

[Kaz05] *KaZaA*. http://www.kazaa.com. Version: 2005

[KBD+06a] KERN, Steffen ; BRAUN, Peter ; DETTBORN, Torsten ; ECKHAUS, Ronny ; JI, Yang ; ERFURTH, Christian ; ROSSAK, Wilhelm: A Generic Agent-based Peer-to-Peer Infrastructure for Social-mobile Applications. In: KIRSTE, Thomas (Hrsg.) ; KÖNIG-RIES, Brigitta (Hrsg.) ; POUSTTCHI, Key (Hrsg.) ; TUROWSKI, Klaus (Hrsg.): *Mobile Informationssysteme - Potentiale, Hinternisse, Einsatz, 1. Fachtagung Mobilität und Mobile Informationssysteme (MMS 2006), Passau (Germany), February 2006* Bd. P-76, Springer Verlag, 2006 (Lecture Notes in Informatics), S. 127–138

[KBD+06b] KERN, Steffen ; BRAUN, Peter ; DETTBORN, Torsten ; ECKHAUS, Ronny ; JI, Yang ; ERFURTH, Christian ; ROSSAK, Wilhelm: Assistant-based Mobile Supply Chain Management. In: RIEBISCH, Matthias (Hrsg.) ; TABELING, Peter (Hrsg.) ; ZORN, Werner (Hrsg.): *13th Annual IEEE International Symposium and Workshops on the Engineering of Computer Based Systems - Mastering the Complexity of Computer-based Systems (ECBS 2006), Potsdam (Germany), March 2006*, IEEE Computer Society Press, 2006, S. 23–31

[KBR06] KERN, Steffen ; BRAUN, Peter ; ROSSAK, Wilhelm: MobiSoft: An Agent-Based Middleware for Social-Mobile Applications. In: MEERSMAN, Robert (Hrsg.) ; TARI, Zahir (Hrsg.) ; HERRERO, Pilar (Hrsg.): *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops, Montpellier, France, October/November 2006, Part II* Bd. 4277, Springer, 2006 (LNCS). – ISBN 3–540–48269–5, S. 984–993

[KHM00] KURATA, Kenji ; HASEGAWA, Go ; MURATA, Masayuki: *Fairness Comparisons between TCP Reno and TCP Vegas – For Future Deployment for TCP Vegas*. http://citeseer.ist.psu.edu/692558.html. Version: 2000

[Kle99] KLEINBERG, Jon: The Small-World Phenomenon: An Algorithmic Perspective / Department of Computer Science, Cornell University. Ithaca NY 14853, October 1999 (Cornell Computer Science Technical Report 99-1776). – Forschungsbericht

[KSHS07] KARNSTEDT, Marcel ; SATTLER, Kai-Uwe ; HAUSWIRTH, Manfred ; SCHMIDT, Roman: Light-weight Internet-scale Universal Storage. In: *Third Biennial Conference on Innovative Data Systems Research*. Asilomar, California, January 7-10 2007

[Kwi07] Kwiatkowski, Lukas: *MCS CPU Benchmark V5, Version 5.03*. http://www.mcsstudios.com/pl/mcscpu.htm. Version: 2007

[KY07] Kanada, Y. ; Yoshioka, A.: *Super Pi, Version 1.5*. http://www.super-computing.org/. Version: 2007

[Lai01] Lai, Ming K.: *Standards for Agents: MASIF and FIPA Specifications*. http://www.cecs.uci.edu/~mingl/agent.html. Version: 2001

[Lim01] Lime Wire LLC: *The Gnutella Protocol Specification v0.4*. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf. Version: 2001

[Lim05] *LimeWire*. http://www.limewire.com. Version: 2005

[Lio04] *Connecting and Extending Peer-to-Peer Networks*. http://lionshare.its.psu.edu/community/lionsharewp.pdf. Version: September 2004

[LSt07a] *Lehrstuhl für Softwaretechnik: Projekt TRACY*. http://swt.informatik.uni-jena.de/pro_tracy.html. Version: April 2007

[LSt07b] *Lehrstuhl für Softwaretechnik: Projekt ProNav*. http://swt.informatik.uni-jena.de/pro_nettracy.html. Version: April 2007

[LSt07c] *Lehrstuhl für Softwaretechnik: Projekt MobiSoft*. http://swt.informatik.uni-jena.de/pro_mobisoft.html. Version: April 2007

[Lö05] Löser, Alexander: *Adaptive Overlays in Peer-to-Peer Netzwerken*, Technische Universität Berlin, Diss., 2005

[Mag05] Magedanz, Thomas: *OMG AND FIPA standardisation for agent technology: competition or convergence?* ftp://ftp.cordis.lu/pub/infowin/docs/omg.pdf. Version: 2005

[Mic03] Microsoft: *Microsoft Windows Server 2003 Active Directory*. http://technet2.microsoft.com/windowsserver/en/technologies/featured/ad/default.mspx. Version: 2003

[Mld05] *MLDonkey*. http://mldonkey.org. Version: 2005

[Moc87a] Mockapetris, P.V.: Domain Names - Concepts and Facilities. (1987). – RFC 1034

[Moc87b] Mockapetris, P.V.: Domain Names - Implementation and Specification. (1987). – RFC 1035

[Mor05] *Morpheus*. http://morpheus.com. Version: 2005

[Neu94] Neuman, B. C.: Scale in Distributed Systems. Version: 1994. http://citeseer.ist.psu.edu/neuman94scale.html. In: Casavant, T. L. (Hrsg.) ; Singhal, M. (Hrsg.): *Readings in Distributed Computing Systems*. Los Alamitos, CA : IEEE Computer Society, 1994, 463–489

[Nov02] Novell Deutschland GmbH: *Novell eDirectory White Paper*. http://www.novell.com/products/edirectory/edirectory_wp.html. Version: 2002

[NSW01] Newman, M. E. J. ; Strogatz, S. H. ; Watts, D. J.: Random graphs with arbitrary degree distributions and their applications. In: *Phys. Rev. E 64:026118* (2001)

[O'R05] O'Reilly, Tim: *What Is Web 2.0 - Design Patterns and Business Models for the Next Generation of Software*. http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html. Version: 2005

[Ora05] Oracle Corporation: *Oracle Application Server 10g Release 2 and 3, New Features Overview, An Oracle White Paper*. http://www.oracle.com/technology/products/ias/pdf/1012_nf_paper.pdf. Version: 2005

[Org04] Organization for the Advancement of Structured Information Standards (OASIS): *UDDI Executive Overview: Enabling Service-Oriented Architecture*. http://uddi.org/pubs/uddi-exec-wp.pdf. Version: 2004

[P+02] Presuhn, R. u. a.: *Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)*. http://tools.ietf.org/html/rfc3418. Version: 2002. – RFC 3418

[Per96] Perkins, C.: *IP Mobility Support*. http://www.faqs.org/rfcs/rfc2002.html. Version: 1996. – RFC 2002

[Per02] Perkins, C.: *IP Mobility Support for IPv4*. http://www.faqs.org/rfcs/rfc3344.html. Version: August 2002. – RFC 3344

[Pos80] Postel, J.: *User Datagram Protocol*. http://www.ietf.org/rfc/rfc0768.txt?number=768. Version: August 1980. – RFC 768

[Pos81a] Postel, J.: *Internet Protocol - DARPA Internet Program - Protocol Specification*. http://www.faqs.org/rfcs/rfc791.html. Version: September 1981. – RFC 791

[Pos81b] Postel, J.: *TRANSMISSION CONTROL PROTOCOL*. http://www.ietf.org/rfc/rfc0793.txt?number=793. Version: September 1981. – RFC 793

[PPW02] Pichler, J. ; Plösch, R. ; Weinreich, R.: MASIF und FIPA: Standards für Agenten Übersicht und Anwendung. In: *Informatik Spektrum, Band 25, Heft 2, April 2002*. Springer Verlag, 2002, S. 91–100

[PR88] Postel, J. ; Reynolds, J.: *A Standard for the Transmission of IP Datagrams over IEEE 802 Networks*. http://www.ietf.org/rfc/rfc1042.txt?number=1042. Version: September 1988. – RFC 1042

[RM90] Rose, M. ; McCloghrie, K.: *Structure and Identification of Management Information for TCP/IP-based Internets*. http://tools.ietf.org/html/rfc1155. Version: 1990. – RFC 1155

[RP99] Rechenberg, P. ; Pomberger, G.: *Informatik-Handbuch*. 2., aktualisierte und erweiterte Auflage. München, Wien : Hanser, 1999

[Sch02] Schmidt, Roman: *Gridella: an open and efficient Gnutella-compatible Peer-to-Peer System based on the P-Grid approach*, Technical University of Vienna, Diplomarbeit, October 2002

[Sch07] Schmidt, Roman: *Publications of Roman Schmidt*. http://lsirpeople.epfl.ch/rschmidt/publications.html. Version: 2007

[SCRT03] Shepler, S. ; Callaghan, B. ; Robinson, D. ; Thurlow, R.: *Network File System (NFS) version 4 Protocol*. http://www.ietf.org/rfc/rfc3530.txt?number=3530. Version: April 2003. – RFC 3530

[Shi01] Shirky, Clay: Listening to Napster. In: Oram, Andy (Hrsg.): *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*. first. O'Reilly and Associates, March 2001, S. 21–37

[SPTU07a] Satzger, B. ; Pietzowski, A. ; Trumler, W. ; Ungerer, T.: A new adaptive accrual failure detector for dependable distributed systems. In: *SAC '07: Proceedings of the 2006 ACM symposium on Applied computing*. New York, 2007

[SPTU07b] Satzger, B. ; Pietzowski, A. ; Trumler, W. ; Ungerer, T.: Variations and Evaluations of an Adaptive Accrual Failure Detector to Enable Self-healing Properties in Distributed Systems. In: *ARCS'07: Architecture of Computing Systems*. Zurich, Switzerland : Springer-Verlag Berlin Heidelberg, March 12 - 15 2007

[Sri95] Srinivasan, R.: *RPC: Remote Procedure Call Protocol Specification Version 2*. http://www.ietf.org/rfc/rfc1831.txt?number=1831. Version: August 1995. – RFC 1831

[Sta07] Standard Performance Evaluation Corporation: *(SPEC)*. http://www.spec.org/. Version: 2007

[Ste04] Stein, Erich: *Taschenbuch Rechnernetze und Internet*. 2. München, Wien : Fachbuchverlag Leipzig im Carl Hanser Verlag, 2004

[Str99] Strassner, John C. ; Technology, MacMillan (Hrsg.): *Directory Enabled Networks*. Indianapolis, U.S. : New Riders Publishing, 1999

[Sun02] Sun Microsystems: *NIS+ to LDAP Migration in the Solaris^{TM} 9 Operating Environment*. http://www.sun.com/software/whitepapers/solaris9/nisldap.pdf. Version: 2002

[Sun03] Sun Microsystems: *Java Native Interface (JNI)*. http://java.sun.com/j2se/1.5.0/docs/guide/jni/spec/jniTOC.html. Version: 2003

[Sun04a] Sun Microsystems: *API of Class Runtime, Java^{TM} 2 Platform Standard Edition 5.0*. http://java.sun.com/j2se/1.5.0/docs/api/index.html. Version: 2004

[Sun04b] Sun Microsystems: *Java^{TM} 2 Platform Standard Edition 5.0, Overview*. http://java.sun.com/j2se/1.5.0/docs/guide/index.html#jre-jdk. Version: 2004

[Sun07a] Sun Microsystems: *Java Management Extensions (JMX)*. http://java.sun.com/j2se/1.5.0/docs/guide/jmx/. Version: 2007

[Sun07b] *Java^{TM} 2 Platform Standard Edition 5.0*. http://java.sun.com/j2se/1.5.0/. Version: 2007

[TA04] Technologies AG ikv++: *Enago Mobile*. http://www.ikv.de/content/Produkte/enago_mobile.htm. Version: 2004

[TAA+03] Traversat, Bernard ; Arora, Ahkil ; Abdelaziz, Mohamed ; Duigou, Mike ; Haywood, Carl ; Hugly, Jean-Christophe ; Pouyoul, Eric ; Yeager, Bill: *Project JXTA 2.0 Super-Peer Virtual Network*. http://www.jxta.org/project/www/docs/JXTA2.0protocols1.pdf. Version: Mai 2003

[Taf06] *the agent factory*. http://www.the-agent-factory.de. Version: 2006

[The00] The Object Management Group (OMG): *Mobile Agent Facility Specification*. http://www.omg.org/technology/documents/formal/mobile_agent_facility.htm. Version: 2000

[The05] The Object Management Group (OMG): *CORBA*. http://www.corba.org/. Version: 2005

[Tra07] Transaction Processing Performance Council: *(TPC)*. http://www.tpc.org/. Version: 2007

[Tri03] Trick, Ulrich: All over IP - Der Schlüssel zur ITK-Infrastruktur der Zukunft. In: *Nachrichtentechnische Zeitschrift* (2003), Nr. 1, S. 30–33

[TS03] Tanenbaum, Andrew S. ; Steen, Maarten van: *Verteilte Systeme - Grundlagen und Paradigmen*. München : Pearson Education Deutschland GmbH, 2003

[Vig98] Vigna, Giovanni: *Mobile Code Technologies, Paradigms, and Applications*, Politecnico di Milano, Diss., Februar 1998

[WHK97] Wahl, M. ; Howes, T. ; Kille, S.: *Lightweight Directory Access Protocol (v3)*. http://www.ietf.org/rfc/rfc2251.txt. Version: December 1997. – RFC 2251

[Wik07a] Wikipedia: *Bootstrapping*. http://de.wikipedia.org/wiki/Bootstrapping. Version: 2007

[Wik07b] Wikipedia: *Carrier Sense Multiple Access Collision Avoidance*. http://de.wikipedia.org/wiki/CSMA/CA. Version: 2007

[Wik07c] Wikipedia: *Failover*. http://de.wikipedia.org/wiki/Failover. Version: 2007

[Wik07d] Wikipedia: *Self-Healing*. http://de.wikipedia.org/wiki/Self_Healing. Version: 2007

[Wik07e] Wikipedia: *Web 2.0*. http://de.wikipedia.org/wiki/Web_2.0. Version: 2007

[Wik07f] Wikipedia: *Wireless Local Area Network*. http://de.wikipedia.org/wiki/Wireless_LAN#Datenraten. Version: 2007

[Xav07] Xavier Gordon: *Hexus PiFast, Version 4.1*. http://numbers.computation.free.fr/Constants/PiProgram/pifast.html. Version: 2007

A Weitere Informationen zu QuickLinkNet

A.1 Weitere Informationen

Die Komponenten des Infrastrukturdienst-Fameworks QuickLinkNet werden ständig verbessert und weiterentwickelt. Der Quellcode ist auf dem Server des Lehrstuhls für Softwaretechnik, Fakultät für Mathematik und Informatik der Friedrich-Schiller-Universität Jena, hinterlegt. Weitere Informationen zum Projekt QuickLinkNet finden Sie auf der Internetseite des Lehrstuhls unter:

http://swt.informatik.uni-jena.de/projekte.html.

A.2 Anhang - QuickLink

A.2.1 Ermittlung der Leistungswerte

In der Komponente QuickLink werden u.a. die Leistungswerte der unterliegenden Java VM bestimmt. Dazu dient das Modul *PerformanceAnalyser*. Die Leistungsermittlung übernimmt dabei die spezielle Klasse `PriorityPerformer`. Sie wird in regelmäßigen Abständen von QuickLink aufgerufen, um die Leistungswerte zu ermitteln.

Zur Leistungsmessung ruft `PriorityPerformer` zuerst das `Benchmark`-Objekt auf, welches die Messung der Rechenleistung übernimmt. Die Ermittlung der Rechenleistung durch `Benchmark` dauert eine gewisse Zeit, in der `PriorityPerformer` abwartet und passiv bleibt. Erst nach der Beendigung der Rechenleistungsbestimmung und der Übernahme des Ergebnisses werden die speicherbezogenen Werte ermittelt. Die Speicherbelegung und -auslastung werden direkt über die Runtime-Methoden der Java VM ausgelesen. Der folgende Code-Ausschnitt verdeutlicht die Arbeitsweise und den Ablauf einer Leistungsmessung in der Klasse `PriorityPerformer`:

```
package QuickLinkNet.PerformanceAnalyzer;
import QuickLinkNet.QuickLink.QuickLink;
  ....

public class PriorityPerformer extends Thread {
  private QuickLink master = null;
  // The to be created Benchmark object
```

```
Benchmark bench = null;
// bench's run state
private boolean benchState = false;
Object synch = null;
private long startTime = 0;
private long perf = 0;
private long memMax = 0;
private long memFree = 0;
private long memTotal =0;
........

public PriorityPerformer(QuickLink master) {
  this.master = master;
  setDaemon(true);
  startTime = System.currentTimeMillis();
  synch = new Object();
  bench = new Benchmark(this);
  start();
}

public void run() {
  while (runState) {
    //Thread pauses until a further notify().
    synchronized (synch){
      try {
        // Wait for "bench" is initialized and
        // remains in the wait-state
        while (bench.getState().compareTo(
          java.lang.Thread.State.WAITING)!= 0) {
          Thread.sleep(4);
          }
          // Remain in a wait-state until a notify
          // wakes up the thread
          synch.wait();
      } catch (InterruptedException e) {e.printStackTrace();}
    }

    // Wake up the performance benchmark
    synchronized(bench) {
      bench.runBench();
    }

    // Wait for benchmark measurement is done ...
    benchState = false;
    while (!benchState) {
      try {
        Thread.sleep(4);
      } catch (InterruptedException e) { e.printStackTrace();}
    }

    // Benchmark is ready, set the result
    perf = bench.getPerformance();
    benchState = false;
    // Get the memory utilization
    memMax = Runtime.getRuntime().maxMemory();
    memFree = Runtime.getRuntime().freeMemory();
    memTotal = Runtime.getRuntime().totalMemory();
    master.setPrioPerfReady(true);
  }
}
........
```

A.2.2 Aufbereitung der Leistungswerte

QuickLink ruft den Thread `PriorityPerformer` alle `<interval>` Sekunden auf und aktualisiert alle Leistungswerte. Nachdem die Leistungswerte ermittelt und QuickLink darüber benachrichtigt wurde, ruft es die Leistungswerte ab rechnet sie in die Performance-Werte um, welche verteilten Anwendungen zur Verfügung gestellt werden. Der folgende Code-Ausschnitt beschreibt diesen Ablauf:

```
package QuickLinkNet.QuickLink;
  ........
public class QuickLink implements Quicklink, Performance, Priorities, Runnable {
  private PriorityPerformer prioPerf = null;
  private int interval;
  ........
  public QuickLink(String[] services) {
    prioPerf = new PriorityPerformer(this);
    ........
    start();
  }
  ........
  public void run() {
    while (runState && !QL.isInterrupted()) {
      // Updates priority values every <interval> seconds.
      if (lastPriorityStamp + interval < System.currentTimeMillis()) {
        getPerformance();
        updateOwnPriorityValues();
      }
      synchronized(QL) {
        try {
          QL.wait(50);
        } catch (InterruptedException e) { e.printStackTrace();}
      }
    }
  }
  ........
  public long getPerformance() {
    return (prioPerf.getPerf());
  }
  ........
  // current maximal amount of memory in Kbytes
  public long getAbsoluteMemoryKb() {return prioPerf.getMemMax()/1024;}
  ........
  // current free amount of memory in Kbytes
  public long getFreeMemoryKb() {return prioPerf.getMemFree()/1024;}
  ........
  // current total amount of memory in Kbytes
  public long getTotalMemoryKb() {return prioPerf.getMemTotal()/1024;}
  ........
  // current memory utilization in %
  public float getMemUtilization() {
      return (prioPerf.getMemMax()-prioPerf.getMemFree())/
                                              ((prioPerf.getMemMax())*100);
  }
  ........
  // current amount of memory used by QuickLinkNet in Kbytes
  public long getQLUsesMemoryKb() {
      return ((prioPerf.getMemTotal() - prioPerf.getMemFree())/1024);
  }
  ........
```

A.2.3 Umrechnung der Leistungswerte in Prioritätswerte

Zum internen Gebrauch, besonders zur Verwendung in den Managementfunktionen ServiceJugglers, verwendet QuickLink Prioritätswerte. Basis für die Prioritätswerte sind die vorhandenen Performance-Werte. Der folgende Code-Ausschnitt zeigt, wie aus ihnen die Prioritätswerte, die in den Wertebereich des Datentyps `Byte` abbilden, berechnet werden. Dabei wird der volle Wertebereich von `Byte`, d.h. auch die negativen Werte von −128 bis −1, nur bei der Rechenleistung und für die Mitgliedszeit eines QuickLink-Knotens im QuickLink-Netzwerk verwendet.

```
package QuickLinkNet.QuickLink;
  ........
  private byte priorityByte = 0;
  private byte freeMemoryMbByte = 0;
  private byte maxMemoryIn10MbAsByte = 0;
  private byte memUtilizationByte = 0;
  private byte upTimeMinByte = 0;
  ........
  private synchronized void updateOwnPriorityValues() {
    // computing power
    setPriorityByte((byte)((getPerformance()/1000000)-128));

    // free memory in Mbytes
    setFreeMemoryMbByte((byte)(getFreeMemoryKb()/1024));

    // max memory in 10 Mbytes
    setMaxMemoryIn10MbAsByte((byte)(getAbsoluteMemoryKb()/10240));

    // memory utilization in %
    setMemUtilizationByte((byte)getMemUtilization());

    // membership in network of this node (stabilization indicator) in minutes
    if ((this.getMyCM().getMemberInNetwork()/60000)>=255) {
      setUpTimeMinByte((byte)(127));
    }
    else
      setUpTimeMinByte((byte)((this.getMyCM().getMemberInNetwork()/60000)-128));
      lastPriorityStamp = System.currentTimeMillis();
    }
  ........
```

Bemerkungen:
Der Divisor für die Berechnung der Rechenleistung (1.000.000) wurde experimentell ermittelt.
Die Mitgliedszeit eines QuickLink-Knotens im QuickLink-Netzwerk wird in Minuten umgerechnet und um -128 im Wertebereich vom Datentyp `Byte` verschoben. Ist der Wertebereich von `Byte` bei 255 Minuten (`Byte`-Wert 127) ausgenutzt, wird die Mitgliedszeit gedeckelt und bei diesem Wert konstant gehalten.

A.2.4 Benchmark

Für den Benchmark wurden Analysekomponenten verwendet, die auf der *Java Grande Forum Benchmark Suite* [EPC07b], *Section 1*, basieren. Sie wurden aber modifiziert, um sie den Einsatzbedingungen in QuickLinks anzupassen. Tabelle A.1 gibt einen Überblick über die verwendeten Klassen und deren Methoden. Für tiefergehende Informationen verweisen wir auf den Quellcode.

Klasse	Methoden
BenchArith	addInt()
	addLong()
	addFloat()
	addDouble()
	multInt()
	multLong()
	multFloat()
	multDouble()
	divInt()
	divLong()
	divFloat()
	divDouble()
BenchAssign	assignSameScalarLocal()
	assignSameScalarInstance()
	assignSameScalarClass()
	assignSameArrayLocal()
	assignSameArrayInstance()
	assignSameArrayClass()
	assignOtherScalarInstance()
	assignOtherScalarClass()
	assignOtherArrayInstance()
	assignOtherArrayClass()
BenchCast	castIntFloat()
	castIntDouble()
	castLongFloat()
	castLongDouble()
BenchCreate	createArrayInt()
	createArrayFloat()
	createArrayObject()
	createObjComplex()
	createObjSubClass()
BenchMethod	methodSameInstance()
	methodSameSynchronizedInstance()
	methodSameFinalInstance()
	methodSameClass()
	methodSameSynchronizedClass()
	methodOtherInstance()
	methodOtherInstanceOfAbstract()
	methodOtherClass()

Tabelle A.1: Die im Benchmark verwendeten Klassen und deren Methoden

A.3 Anhang - ServiceJuggler

Da bei der Untersuchung von fertigen LDAP-Lösungen keine geeignete Implementierung gefunden werden konnte, wurde ein eigener Verzeichnisdienst im ServiceRegistry ServiceJugglers implementiert. Er verarbeitet LDAP-konforme Anfragen und wurde vollständig in Java implementiert, um eine gute Migrationsfähigkeit sicherzustellen. Das folgende Bild zeigt das Klassendiagramm des Verzeichnisdienstes und gibt einen Überblick über dessen objektorientierte Struktur.

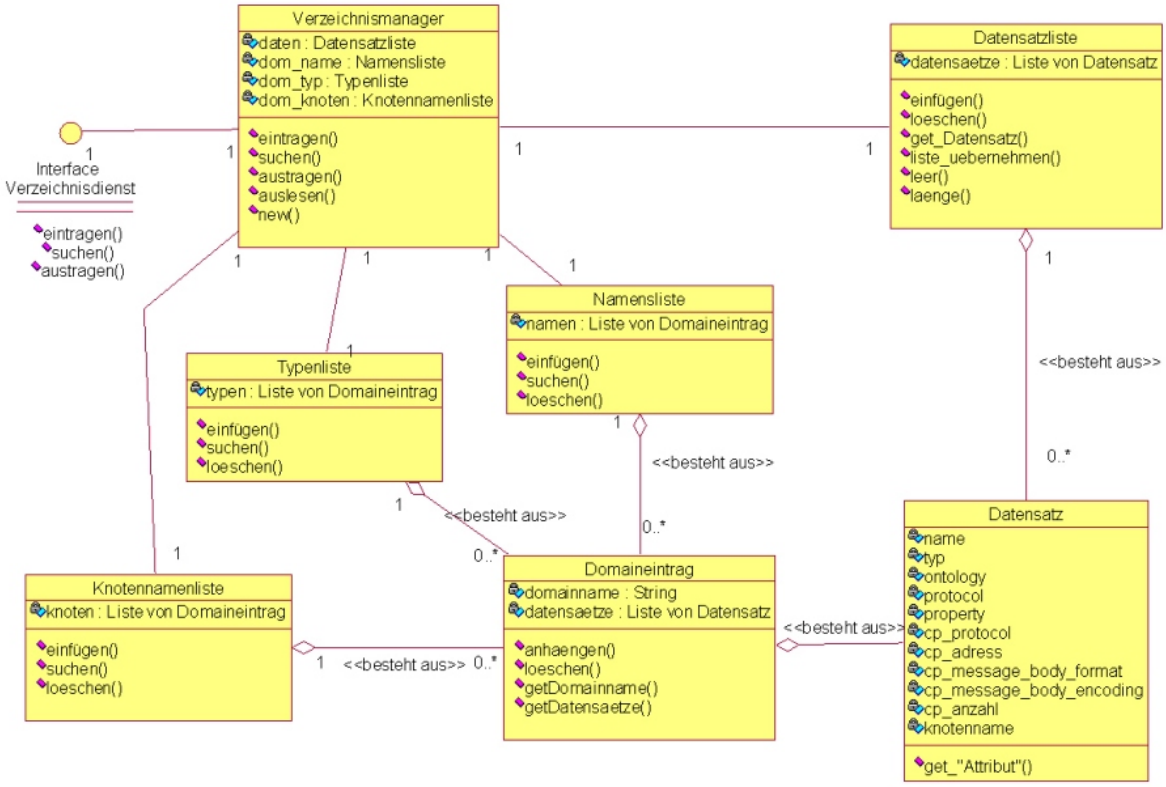

Abbildung A.1: Das Klassendiagramm des Verzeichnisdienstes des ServiceRegistry

A.4 Anhang - APLICOOVER

Der Test APLICOOVERs erfolgte über ein Testskript, welches die folgend aufgelisteten Methoden in der angegebenen Reihenfolge aufrief:

```
...
adminStub = (remote.RemoteAdministration) registry.lookup(”APLICOOVER”);
standardStub = (remote.RemoteServices) registry.lookup(”APLICOOVER”);
...
//Bootstrap
adminStub.init();
...
//Publish region profile
String[] bundleDescription = adminStub.publishRegionProfile();
...
//Searching
String queryGUID = standardStub.query(”Testdienst”);
...
//Searching
queryGUID = standardStub.query(”Testdienst1_192.168.2.2”);
...
//Modify first service
String guidOfModifiedService = utils.Utility.extractGUID(bundleDescription[0]);
standardStub.updateService(guidOfModifiedService, ”ModifizierterTestdienst1”);
...
//Delete second service
String guidOfDeletedService = utils.Utility.extractGUID(bundleDescription[1]);
standardStub.deleteService(guidOfDeletedService);
...
//Searching for modified service
queryGUID = standardStub.query(”Modifizierter”);
...
//Searching for deleted service
queryGUID = standardStub.query(”Testdienst2_192.168.2.21”);
...
//Range search
queryGUID = standardStub.rangeQuery(”M”,”T”);
...
//Cached search, non globally, cache-hit
//to-be-searched information is deleted in the p2p layer first
//to make sure the results come from cache
```

```
standardStub.deleteService(guidOfModifiedService);
...
queryGUID = standardStub.cachedQuery(”Modifizierter”, false);
...
//Cached range search - globally
queryGUID = standardStub.cachedRangeQuery(”Mo”,”Te”,true);
...
//shutdown
adminStub.shutdown();
```

Folgende Ausgaben der Testumgebung (in komprimierter Form) entstanden beim Funktionstest APLICOOVERs. Sie stellen die Antworten des Systems auf die Aufrufe aus dem auf Seite 280 beschriebenen Testskript dar:

```
Launching bootstrap...done.
Publishing service descriptions......done.
...

Searching for 'Testdienst'...
Testdienst1_192.168.2.21_#1@192.168.2.21
Testdienst2_192.168.2.21_#2@192.168.2.21
Testdienst3_192.168.2.21_#3@192.168.2.21
Testdienst4_192.168.2.21_#4@192.168.2.21
Testdienst3_192.168.2.20_#3@192.168.2.20
Testdienst4_192.168.2.20_#4@192.168.2.20
Testdienst2_192.168.2.20_#2@192.168.2.20
Testdienst1_192.168.2.20_#1@192.168.2.20
Testdienst1_192.168.2.16_#1@192.168.2.16
Testdienst3_192.168.2.16_#3@192.168.2.16
Testdienst4_192.168.2.16_#4@192.168.2.16
Testdienst2_192.168.2.16_#2@192.168.2.16
Testdienst2_192.168.2.19_#2@192.168.2.19
Testdienst1_192.168.2.19_#1@192.168.2.19
Testdienst3_192.168.2.19_#3@192.168.2.19
Testdienst4_192.168.2.19_#4@192.168.2.19

Searching for 'Testdienst1_192.168.2.2'...
Testdienst1_192.168.2.21_#1@192.168.2.21
Testdienst1_192.168.2.20_#1@192.168.2.20

Updating first published service...
Deleting second published service...
Searching for 'Modifizierter'...
ModifizierterTestdienst1@192.168.2.21
ModifizierterTestdienst1@192.168.2.16
ModifizierterTestdienst1@192.168.2.20
ModifizierterTestdienst1@192.168.2.19

Searching for deleted service...

Range search for 'M'-'T'...
```

```
ModifizierterTestdienst1@192.168.2.21
Testdienst3_192.168.2.21_#3@192.168.2.21
Testdienst4_192.168.2.21_#4@192.168.2.21
Testdienst3_192.168.2.19_#3@192.168.2.19
Testdienst4_192.168.2.19_#4@192.168.2.19
Testdienst4_192.168.2.16_#4@192.168.2.16
Testdienst4_192.168.2.20_#4@192.168.2.20
Testdienst3_192.168.2.20_#3@192.168.2.20
Testdienst3_192.168.2.16_#3@192.168.2.16
ModifizierterTestdienst1@192.168.2.20
ModifizierterTestdienst1@192.168.2.16
ModifizierterTestdienst1@192.168.2.19

Deleting modified service...
Cached search (not globally cont'd) for 'Modifizierter'...
Found in cache:
ModifizierterTestdienst1@192.168.2.21
ModifizierterTestdienst1@192.168.2.20
ModifizierterTestdienst1@192.168.2.16
ModifizierterTestdienst1@192.168.2.19

Cached range search (globally cont'd) for 'Mo'-'Te'...
Not found in cache but in the p2p layer:
ModifizierterTestdienst1@192.168.2.21
Testdienst3_192.168.2.21_#3@192.168.2.21
Testdienst4_192.168.2.21_#4@192.168.2.21
Testdienst4_192.168.2.16_#4@192.168.2.16
Testdienst4_192.168.2.20_#4@192.168.2.20
Testdienst3_192.168.2.16_#3@192.168.2.16
Testdienst3_192.168.2.20_#3@192.168.2.20
Testdienst3_192.168.2.19_#3@192.168.2.19
Testdienst4_192.168.2.19_#4@192.168.2.19
ModifizierterTestdienst1@192.168.2.16
ModifizierterTestdienst1@192.168.2.20
ModifizierterTestdienst1@192.168.2.19

Shutting down APLICOOVER instance...
```

Selbständigkeitserklärung:
Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Hilfsmittel und Literatur angefertigt habe.

Jena, den 10. Dezember 2007

.....

Lebenslauf

Name:	Arndt Döhler
Geburtsdatum:	13.01.1971
Geburtsort:	Apolda
Anschrift:	Hausbergstraße 25 07749 Jena
Schulbildung:	1977 - 1987 POS "Friedrich-Engels", Apolda
Berufsausbildung:	1987 - 1990 Elektronikfacharbeiter, Fachrichtung "Instandhaltung" Feuerlöschgerätewerk Apolda / VEB Mikroelektronik "Karl Marx" Erfurt
Wehrersatzdienst:	Mai 1992 - Juli 1993
Technikerausbildung:	1993 - 1995 Staatlich geprüfter Elektrotechniker (1995) Erwerb der Fachhochschulreife (1995) Fachschulen Hermsdorf und Gotha
Studium:	1995 - 2000 Wirtschaftingenieurwesen Fachhochschule Jena Vertiefungsrichtungen: Informations- und Kommunikationstechnik, Produktionsmanagement Abschluss Diplom, Dipl.-Wirt.-Ing. (FH) (2000) Diplomarbeit "Einsetzbarkeit eines Elektroluftfilters im KFZ" bei der BMW M GmbH, daraus resultierend Miterfinder in den Europäischen Patentanmeldungen 00121830.4-2313 und 00121829.6-2313
Wissenschaftliche Tätigkeit:	März 2001 - Mai 2004 FH Jena, wissenschaftlicher Mitarbeiter diverse Lehraufträge seit Oktober 2004 FSU Jena, wissenschaftlicher Mitarbeiter

Jena, den 10. Dezember 2007